VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

DIPLOMOVÁ PRÁCE

2012 Bc. Miroslav Tichý

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra Informatiky a výpočetní techniky

Měření akustické emise

Acoustic emission measurement

2012 Bc. Miroslav Tichý

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student:	**Bc. Miroslav Tichý**
Studijní program:	N2647 Informační a komunikační technologie
Studijní obor:	2612T025 Informatika a výpočetní technika
Téma:	Měření akustické emise Acoustick Emission Measurement

Zásady pro vypracování:

Návrh a realizace zařízení pro měření akustických emisí na bázi hradlových polí Actel s procesorovým jádrem s Ethernet rozhraním pro komunikaci s nadřízeným systémem.

1. Seznámit se s principem metody měření akustické emise.
2. Seznámit se s hradlovými poli Actel.
3. Navrhnout a realizovat technické vybavení (hardware) a jeho připojení k jádru procesoru včetně simulací.
4. Navrhnout a odladit programové vybavení (firmware).

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jiří Mitrych, Ph.D.**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka	prof. RNDr. Václav Snášel, CSc.
vedoucí katedry	*děkan fakulty*

Prohlášení:

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 3.5.2012 …….

Bc. Miroslav Tichý

Poděkování:

Chtěl bych poděkovat panu Ing. Jiřímu Mitrychovi Ph.D za odborné a organizační vedení při zpracování této práce.

Abstrakt

Cílem této diplomové práce je seznámit se s principem metody akustické emise, která se používá pro nalezení možných vad materiálu, vznikajících namáháním, a navrhnout a realizovat zařízení pro její měření. Zařízení bude pracovat na bázi programovatelných hradlových polí Actel s procesorovým jádrem a s Ethernet rozhraním pro komunikaci s nadřízeným systémem. K realizaci bylo vybráno hradlové pole rodiny SmartFusion ve spolupráci s procesorem Cortex-M3. Pro implementaci logických obvodů do tohoto pole bylo využito vývojové prostředí Libero IDE. Toto prostředí, jehož součástí je i simulátor ModelSim, pomocí kterého si ověříme funkčnost návrhu, poskytuje rovněž firma Actel. Pro popis hardwarových struktur hradlových polí byl použit programovací jazyk VHDL a pro naprogramování procesoru programovací jazyk C a vývojové prostředí SoftConsole IDE. Věřím, že má diplomová práce splní svůj účel, osvětlí problematiku akustické emise a srozumitelně vysvětlí mnou zvolený způsob realizace tohoto systému.

Klíčová slova:

Akustická emise, Actel, Ethernet rozhraní, FPGA, LiberoIDE, SmartFusion, SoftConsoleIDE

Abstract

The aim of this thesis is to introduce the principle of the method of acoustic emission and to design facilities for its measurement. This method is used for discovering potential material defects caused by stressing it. The device will operate on the basis of Actel programmable gate arrays with processor core and Ethernet interface for communication with host system. Gate array of SmartFusion family in collaboration with Cortex-M3 processor was chosen for the implementation. Actel company also provides the development environment Libero IDE which was used to implement the logic circuits into this array. This development environment includes the Model Sim simulator which enables the design functionality testing. Two programming languages were used: VHDL for the description of the hardware structure and C with SoftConsole IDE for CPU programming. I believe that this thesis has fulfilled its purpose, clarifies issues of acoustic emission and clearly explains my chosen method of the system implementation.

Keywords:

Acoustic emission, Actel, Ethernet interface, FPGA, LiberoIDE, SmartFusion, SoftConsoleIDE

Seznam použitých symbolů a zkratek

ACE	Analog Compute Engine
AE	Akustická emise
AES	Advanced Encryption Standard
AHP	Advanced High-performance Bus
AMBA	Advanced Microprocessor Bus Architecture
APB	Advanced Peripheral Bus
API	Application Programming Interface
ASIC	Application-specific integrated circuit
CPLD	Complex Programmable Logic Device
DLL	Delay Locked Loop
FIC	Fabric interface controller
FPGA	Field Programmable Gate Array
IDE	Integrated Design Environment
LAN	Local Area Network
LUT	Look-up Table
MAC	Media Access Control
MSS	Microcontroller Subsystem
PHY	Physical Layer Interface
PLD	Programmable Logic Device
PLL	Phase Locked Loop
RMS	Root Mean Square
SRAM	Static Random Access Memory
UART	Universal asynchronous receiver transmitter
UUT	Unit Under Test
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuit

Obsah

1. Úvod ..... - 4 -
2. Akustická emise ..... - 5 -
2.1 Akustická emise ..... - 5 -
2.1.1 Emisní událost ..... - 5 -
2.1.2 Emisní zdroj ..... - 5 -
2.2 Princip akustické emise ..... - 6 -
2.3 Zpracování signálu ..... - 7 -
2.3.1 Forma zpracování signálů ..... - 7 -
2.3.2 Popis signálu pomocí základních parametrů akustické emise ..... - 7 -
2.3.3 Hodnocení výsledků měření akustické emise ..... - 9 -
2.4 Výhody a nevýhody použití metody akustické emise ..... - 10 -
2.5 Aplikační možnosti metody ..... - 10 -
3. Programovatelná hradlová pole ..... - 11 -
3.1 Základní pojmy ..... - 11 -
3.2 FPGA ..... - 11 -
3.2.1 Historický vývoj FPGA ..... - 11 -
3.2.2 Základní vlastnosti FPGA ..... - 12 -
3.2.3 Architektura FPGA ..... - 12 -
3.2.4 Uplatnění hradlových polí FPGA ..... - 15 -
4. Programovatelné hradlové pole rodiny SmartFusion ..... - 16 -
4.1 MSS ..... - 17 -
4.2 Analogová část ..... - 17 -
4.3 Programovatelné hradlové pole ..... - 17 -
4.4 SmartFusion Evaluation Kit ..... - 18 -
5. Jazyk VHDL ..... - 20 -
6. Systémový návrh ..... - 22 -
6.1 Systémové požadavky ..... - 22 -
7. Hardwarový návrh ..... - 25 -
7.1 Čítače a jejich návrh ..... - 25 -
7.1.1 Čítače ..... - 25 -
7.1.2 Návrh čítače ..... - 27 -
7.2 Návrh měřicího obvodu ..... - 28 -
7.2.1 Zapojení vstupních signálů čítače ..... - 28 -
7.2.2 Přivedení dat na sběrnici ..... - 29 -
7.2.3 Obecná charakteristika APB sběrnice ..... - 30 -
7.2.4 Můstky mezi MSS-APB-Periferií ..... - 31 -
7.2.5 Dokončení přivedení dat na APB ..... - 32 -
7.3 Konfigurace MSS ..... - 34 -
7.3.1 Clock management ..... - 36 -
7.3.2 Reset Management ..... - 36 -
7.3.3 Fabric Interface ..... - 36 -

7.3.4 MAC ..... - 37 -
7.4 Analýza čítací frekvence ..... - 37 -
7.5 Simulace ..... - 38 -
7.5.1 Testbench ..... - 39 -
7.5.2 Simulace měřícího obvodu ..... - 41 -
7.6 Časové parametry číslicových obvodů ..... - 43 -
7.6.1 Časová analýza ..... - 44 -
7.7 Ethernet rozhraní ..... - 46 -
7.8 Programové vybavení pro návrh FPGA obvodů ..... - 47 -
7.8.1 Postup tvorby návrhu v prostředí Libero IDE ..... - 47 -
7.8.1.1 Design Entry Tools (BET) ..... - 49 -
7.8.1.2 Synthesis (Syntéza) ..... - 49 -
7.8.1.3 Place & Route ..... - 50 -
7.8.1.4 Programming ..... - 50 -
7.8.1.5 Simulation ..... - 50 -
8. Softwarový návrh ..... - 51 -
8.1 SoftConsole IDE ..... - 51 -
8.2 Tvorba programu ..... - 51 -
9. Závěr ..... - 55 -
10. Literatura ..... - 56 -
Obsah přiloženého CD

Seznam obrázků

***Obrázek 2.1**:Princip AE [3]*……..…- 6 -
***Obrázek 2.2**: Schéma měřící trasy...*……..…….. - 7 -
***Obrázek 2.3**: Průběh signálu AE [2]*…….…… - 8 -
***Obrázek 2.4**: Spojitý signál (vlevo) a nespojitý praskavý signál (vpravo) AE [4]*……..- 9 -
***Obrázek 3.1**: Struktura obvodu FPGA [6]*……..…- 13 -
***Obrázek 3.2**: Struktura konfigurovatelného logického bloku [6]*……...…… - 13 -
***Obrázek 4.1**: Schéma SmartFusion zařízení [10]*……... - 16 -
***Obrázek 4.2**: SmartFusion Evaluation Kit (A2F-EVAL-KIT)[11]*……..- 18 -
***Obrázek 6.1**: Blokové schéma navrhovaného systému*……....- 24 -
***Obrázek 7.1:** Ukázka sestavení asynchronního čítače s KO typu D* ……..……..... - 26 -
***Obrázek 7.2**: Část VHDL kódu čítače* ……..……... - 27 -
***Obrázek 7.3**: Blok navrženého čítače* …….…… - 28 -
***Obrázek 7.4**: Část schématu zapojení*…….…… - 29 -
***Obrázek 7.5**: Typický AMBA systém [12]*.....……..... - 31 -
***Obrázek 7.6**: Schéma zapojení bloku hradlového pole do MSS [12]*…….....- 31 -
***Obrázek 7.7**:Blokové schéma komponenty CoreAPB3 [13]*……....- 33 -
***Obrázek 7.8**: Schéma MSS+APB+CITAC*…….... - 34 -
***Obrázek 7.9**: Mikrokontrolér subsystém (MSS)*…….…… - 35 -
***Obrázek 7.10**: Postup Simulace* …….. - 39 -
***Obrázek 7.11**: Testbench – struktura* ……..…… - 40 -
***Obrázek 7.12**: Časový diagram měřícího obvodu* ……..…….- 42 -
***Obrázek 7.13**: Část tabulky zpoždění pro signál CLK* ……..……. - 44 -
***Obrázek 7.14**: Průchod CLK částí obvodu s největším zpožděním* ……...……- 45 -
***Obrázek 7.15**: Max. frekvence a Min. perioda* ……...…- 45 -
***Obrázek 7.16**: Poměr zpoždění v buňkách FPGA a jejich přechodech*…….... - 45 -
***Obrázek 7.17**: Kroky postupu tvorby programovatelného obvodu*…….. - 48 -

1. Úvod

Lidé již od pradávna věděli, že v každém tělese, které je vystaveno působení určitých vlivů (tlak, teplota, povětrnostní vlivy apod.) dochází postupně ke změnám jeho kvality. Jako nejjednodušší metodu k posouzení této změny používali poklepání na těleso a poslouchali odezvu na tento úder, ze které pak usuzovali na případnou skrytou vadu. Postupně si však uvědomovali, že je třeba zdokonalovat si vědomosti o vlastnostech a chování materiálů a tak především proto, aby se předešlo různým haváriím a katastrofám, vznikl v průběhu 19. století technický obor zkušebnictví, který pomocí různých metod odhaluje vady a odlišnosti ve struktuře materiálů. Tento obor se od poklepu na těleso postupně vyvíjel a současné akustické metody testování se zakládají na stále přesnějších postupech měření, především co se týká buzení a příjmu zvuku.

Jednou z pasívních inkoherentních metod, která využívá k měření změn postupných vlnových pulsů, je metoda akustické emise a cílem mé diplomové práce je navrhnou a realizovat zařízení pro měření akustických emisí na bázi hradlových polí Actel s procesorovým jádrem a s Ethernet rozhraním pro komunikaci s nadřízeným systémem.

Druhou kapitolu jsem tedy věnoval akustické emisi a třetí hradlovým polím FPGA, jejich charakteristice a vlastnostem. Ve čtvrté části jsem popsal integrovaný obvod FPGA rodiny SmartFusion od firmy Actel a zmínil jsem se o prostředí Libero IDE, ve kterém jsou aplikace hradlových polí navrhovány. Pátá kapitola stručně charakterizuje jazyk VHDL, který byl použit pro popis hardwarových struktur hradlových polí. Pak jsem přešel k samotné realizaci návrhu mého systému a v následujících třech kapitolách jsem vytvořil a popsal systémový, hardwarový a softwarový návrh. Závěrečná devátá kapitola je stručným shrnutím celé mé práce.

2. Akustická emise

Protože zadání práce si klade za cíl návrh systému pro měření akustické emise, povíme si v této kapitole nejprve něco blíže o tomto pojmu a o stejnojmenné metodě měření vad v materiálu s tím související. Kromě základních definic si popíšeme její princip, zpracování, vyhodnocování, možnosti využití a další.

2.1 Akustická emise

Akustickou emisí je nazýváno prudké uvolňování energie z lokálních zdrojů v materiálu za současného působení stimulů, jakými jsou například tlak, teplota a vnitřní pnutí během provozu exponovaných těles. Jde vlastně o fyzikální jev, při němž pozorujeme akustické signály a také je tak nazývána diagnostická metoda, která je na tomto jevu založena.

Uvolněná energie se přeměňuje na impuls napětí, který se v momentě, kdy dosáhne povrchu materiálu, začne šířit formou elastických vln a ty je pak možno detekovat pomocí speciálně konstruovaných, piezoelektrických snímačů akustických emisí.

2.1.1 Emisní událost

Emisní událost je jednorázový dynamický proces, jehož důsledkem dojde k rychlému uvolnění určitého množství energie. Jednotlivé emisní události jsou různě dlouhé (již od několika nanosekund) a jejich frekvenční spektrum je široké od infrazvukových do ultrazvukových frekvencí.

2.1.2 Emisní zdroj

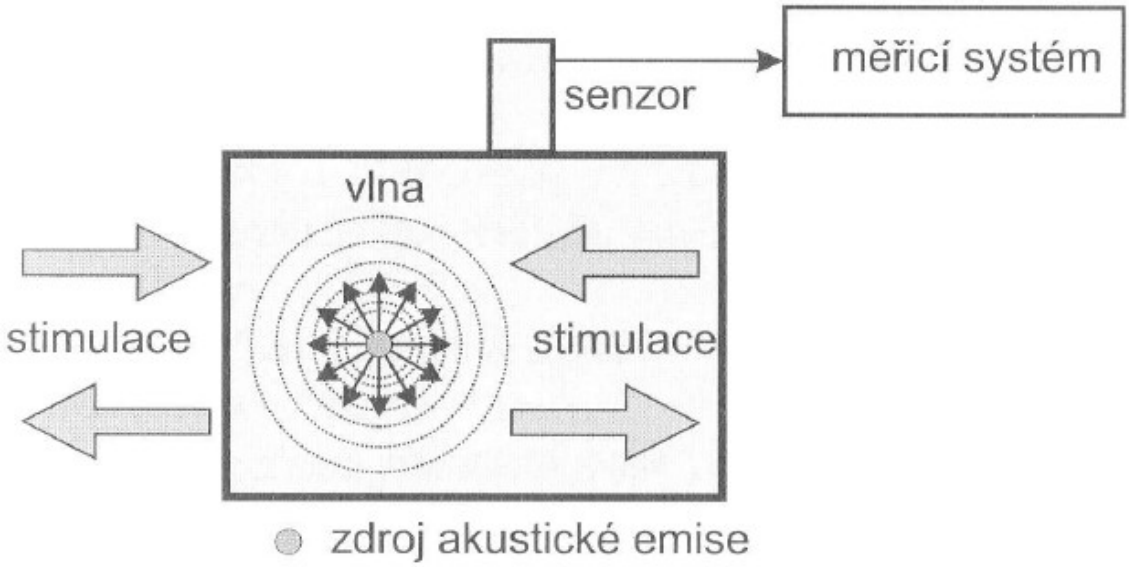
Emisní zdroj je lokální proces, který produkuje emisní událost. Jak je uvedeno výše, k uvolnění akumulované energie v materiálech je třeba určitého vnějšího nebo vnitřního podnětu a právě zdrojem elastických vln může být řada fyzikálně metalurgických procesů jako: [5]

- Plastická deformace
- Vznik a přerozdělování pnutí
- Korozní praskání
- Vznik a šíření vad v materiálu
- Úniky
- Výrobní technologie

2.2 Princip akustické emise

Metoda AE pracuje na základě "odposlechu" varovných signálů, které vznikají a rozvíjejí při zatěžování konstrukce. Tato metoda odkrývá akusticky aktivní defekty, které se stávají aktivními díky řadě příčin, jako je např. koroze, vysoké namáhání, oslabená struktura materiálu apod.

Metoda AE je integrální. Síť snímačů umožňuje monitorovat zařízení přímo při zatěžování, a to i v provozu. Přitom detektory nás informují nejen o přítomnosti defektu, ale i o podmínkách pro jeho rozvoj. Princip AE je patrný z ***obrázku 2.1***.



Obrázek 2.1: *Princip AE* [3]

Při zatěžování struktury vzniká trhlina, která je vlastně zdrojem elastické vlny. Předpokládá se, že tato vlna je kulová, a šíří se do všech směrů stejnou rychlostí. V momentě, kdy se dostane na povrch materiálu, vytvoří povrchovou vlnu, která je zachycena sítí senzorů umístěnou na povrchu tělesa. Tyto snímače jsou nejčastěji širokopásmové (od 100 kHz do 4 MHz) piezoelektrické senzory, jejichž rezonanční frekvence je nad měřeným spektrem akustické emise nebo citlivější rezonanční senzory s více rezonancemi. Ty detekují složku elastického vlnění kolmou k povrchu konstrukce a transformují ji na nízkonapěťový elektrický signál. Moderní detektory mají poblíž snímače umístěn předzesilovač, který signál zesiluje a filtruje, čímž se vyloučí nízkofrekvenční mechanické a elektrické rušení. Signál je opět snížen a přes koaxiální kabel jde k měřicímu systému, kde je znovu zesílen. Měřící proces začne v momentě, kdy hodnota zesíleného a filtrovaného signálu vystoupí nad určenou prahovou úroveň. Měření jednoho balíku AE bude ukončeno v momentě, kdy po určitou, námi zvolenou dobu, měřený

signál nevystoupí nad úroveň prahu. Tento balík AE je pak označen a zařízení může přijmout balík nový. Zpravidla až po ukončení měření, se provádí analýza a zobrazení signálu. Schéma měřicí trasy ukazuje ***obrázek 2.2***. [1]

***Obrázek 2.2**: Schéma měřící trasy*

2.3 Zpracování signálu

2.3.1 Forma zpracování signálů

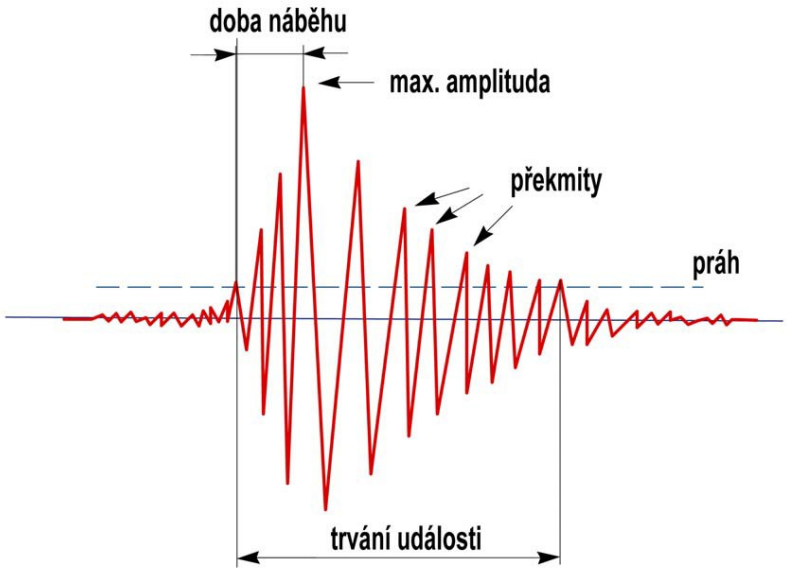
Zpracování signálu se provádí formou:

- Registrace a lokalizace zdrojů emisních událostí s následným vyhodnocováním jejich závažností
- Vyhodnocováním emisních překmitů v případě spojitého signálu bez výrazných vzplanutí
- Kombinací obou způsobů zpracování [5]

2.3.2 Popis signálu pomocí základních parametrů akustické emise

Při sledování průběhu AE si nejdříve stanovíme prahovou hodnotu a pak sledujeme, kdy signál tuto stanovenou hodnotu překročí. Doba od dosažení a překročení prahové hodnoty až do doby, kdy ji signál znovu nepřekročí, se nazývá emisní událost.

Na ***obrázku 2.3*** jsou znázorněny některé parametry, pomocí kterých si můžeme emisní událost popsat.



***Obrázek 2.3**: Průběh signálu AE* [2]

- **Práh** je námi nastavená hodnota, která definuje začátek a konec jednotlivých emisních událostí.
- **Počet emisních překmitů** – udává počet kmitů, které vystoupí nad nastavený práh. Každá emisní událost je zpravidla zaznamenána velkým počtem překmitů.
- **Amplituda emisní události** – je maximální výchylka emisní události.
- **Doba trvání emisní události** – je doba, která uběhne mezi prvním a posledním překročením hodnoty prahu
- **Doba náběhu** – je čas mezi prvním překročením prahu a maximální výchylkou. Využívá se k rozlišení zdroje signálu a jeho filtraci.
- **Průměrná frekvence** – poměr mezi množstvím emisních překmitů a časem trvání emisní události.
- **Čas příchodu** – je čas, kdy došlo k překročení prahové hodnoty.
- **Časový rozdíl** – jde o časovou diferenci téže emisní události jiným snímačem. Tento parametr může být využit pro lokalizaci zdroje AE.

2.3.3 Hodnocení výsledků měření akustické emise

Základní vyhodnocení signálu provádí měřící systém AE. Ten detekuje jednotlivé hity (časově oddělené pulsy) elektrického signálu, vzniklého transformací rázových vln. Tyto hity tvoří buď jednoduchou posloupnost, nebo překrývající se posloupnost.

Podle charakteru elektrického signálu tedy rozlišujeme dva druhy AE a to:

- **Emise spojitá** (překrývající se posloupnost), je vyvolána událostmi, které nejsou časově odděleny, to znamená, že u ní po delší dobu nedojde k poklesu signálu pod nastavenou prahovou hodnotu. Příčinou vzniku tohoto signálu jsou zpravidla různé fyzikální procesy, jako např. kavitace v kapalině, plastická deformace kovů vzniklá při tření, únik kapaliny pod tlakem z nádoby nebo potrubí apod.

- **Emise nespojitá** (jednoduchá posloupnost), kdy signál má charakter jednotlivých, časově výrazně oddělených balíků. Jejím typickým příkladem je např. vznik a růst trhliny, klepání uvolněných částí apod.

***Obrázek 2.4**: Spojitý signál (vlevo) a nespojitý praskavý signál (vpravo) AE* [4]

Konkrétním výsledkem metody AE v případě provozních kontrol je klasifikace jednotlivých zdrojů podle stupně závažnosti, kterou upravuje norma ASTM E596-91. Ta rozděluje emisní zdroje podle stupně závažnosti na zdroje:

- **Nezávažné (neaktivní),** z jejichž příznaků usuzujeme na nezávažnost defektu. U takovýchto zdrojů není potřeba provádět další defektoskopické kontroly, pouze se bude provádět opakované měření, které bude sledovat jejich případný rozvoj.

- **Potenciálně závažné (aktivní)**, jejichž příznaky odpovídají defektu, který sice bezprostředně neohrožuje provoz zařízení, je však závažný z hlediska budoucího provozu zařízení, kdy vlivem reálných zatěžujících parametrů může dojít k jeho šíření a tím je nepříznivě ovlivněna doba životnosti zařízení. Zdroje tohoto typu je třeba klasifikovat dodatečnými defektoskopickými kontrolami, případně pomocí AE sledovat jejich rozvoj.

- **Kriticky závažné (kriticky aktivní)** jsou zdroje, které podle příznaků bezprostředně ohrožujícího provoz zařízení a při jejich zjištění je tedy nutné zařízení okamžitě odstavit.

2.4 Výhody a nevýhody použití metody akustické emise

Akustická emise je jednou z univerzálních metod nedestruktivní diagnostiky. Její nespornou výhodou je, že je metodou pasivní, tzn., že měřený objekt neovlivňuje, pouze detekuje přítomnost defektu a potenciálně nebezpečné podmínky pro jeho rozvoj. Je vysoce citlivá k přechodovým lokálním nestabilitám, které vznikají v materiálu mnohem dříve, než se nestabilní stane celá konstrukce. Jejich včasným zachycením lze tedy mnohdy předejít poruše zařízení.

Dalším plusem je, že k monitoringu celého kontrolovaného objektů stačí jen malý počet snímačů a taky to, že nám umožňuje kontrolovat obtížně přístupná místa a odhalovat i trhliny, které jsou tak malé, že je není možno zachytit jinými metodami.

Její nevýhodou je vysoká cena měřicího zařízení a složité vyhodnocování změřených výsledků.

2.5 Aplikační možnosti metody

V praxi se akustická emise používá nejčastěji při řešení dvou úloh - monitorování úniků a detekce trhlin a umožní nám provádět zejména tyto kontroly: [5]

1. **Kontrola výrobní technologie**
   - svařování
   - tepelné zpracování
   - tuhnutí odlitků
2. **Kontrola technického stavu konstrukcí**
   - výstupní tlakové zkoušky
   - periodické tlakové zkoušky
3. **Trvalá diagnostika za provozu**
   - změny provozních parametrů
   - vznik pnutí a deformačního namáhání
   - vznik a šíření defektů
   - včasná indikace úniků pracovních médií

Z výše uvedeného vyplývá, že tato metoda najde uplatnění v řadě průmyslových odvětví jako například při kontrole tlakových nádob a potrubí, tlakových zásobníků, reaktorů, autoklávů, cisteren, kotlů, tepelných výměníků a jiných.

3. Programovatelná hradlová pole

S rozvojem a modernizací elektrotechnického průmyslu se stále zvyšují nároky na výpočetní výkon, kompaktnost řešení, rekonfigurovatelnost systému a taky rostou požadavky na snižování ekonomických nákladů, tzn., že je požadována minimální cena za co nejkratší vývojový cyklus produktu.

Řešení nabízí programovatelná logika a zejména hradlová pole. Jedná se o programovatelnou součástku, pomocí které si můžeme s minimálními náklady vytvořit vlastní integrovaný obvod, přizpůsobený konkrétní aplikaci.

3.1 Základní pojmy

Všechny číslicové programovatelné součástky označujeme názvem PLD (Programmable Logic Device) a podle vnitřní struktury je můžeme rozdělit na:

- **Klasické PLD –** jednoduché obvody, které díky velmi omezeným prostředkům umožňují realizovat jen jednodušší funkce.
- **Komplexní PLD –** označují se většinou jako CPLD (Complex Programmable Logic Device) a sdružují na jednom čipu více obvodů spolu s prostředky nutnými pro propojení.
- **Obvody typu FPGA** (Field Programmable Gate Array) - mají z programovatelných obvodů nejobecnější strukturu a obsahují nejvíce logiky.

3.2 FPGA

Programovatelná hradlová pole (**FPGA** - Field Programmable Gate Array) jsou speciální číslicové integrované obvody obsahující různě složité programovatelné bloky propojené konfigurovatelnou maticí spojů. Field Programmable v názvu je to, čím se FPGA odlišují od zákaznických integrovaných obvodů – obvod je „perzonifikován“ (nakonfigurován) u zákazníka. Současné největší obvody FPGA obsahují až 6 milionů ekvivalentních hradel (typické dvouvstupové hradlo NAND).[8]

3.2.1 Historický vývoj FPGA

Historické kořeny moderních programovatelných polí jsou v prvních programovatelných pamětech typu PROM (firma Radiation, 1970) a jejich zákaznicky programovatelných verzím EPROM (Intel, 1971) a EEPROM (Intel, 1978). Permanentní paměti jako takové ale neumožňovaly úspornou realizaci logické funkce. Logickým vývojovým krokem proto byl vznik tzv. FPLA obvodů (Field Programmable Logic Array, Signetics/Philips, 1970), ty ale byly

drahé a pomalé. Aby bylo možné dosáhnout vyšší rychlosti logiky, byla součtová matice realizována jako fixní - vznikla koncepce PAL obvodů (Programmable Array Logic, Monolithic Memories, 1978) s programovatelnou maticí AND a pevnou maticí OR. S postupně se zlepšujícími technologiemi výroby integrovaných obvodů bylo možné vyrábět programovatelné obvody s vyšší kapacitou a realizující složitější logické funkce. Kvůli rostoucí velikosti obvodů se začalo později místo rozšiřování logických funkcí užívat spíše skládání více matic PLD obvodů do jednoho pouzdra. Vznikly tak obvody, které dnes nazýváme CPLD (Complex Programmable Logic Device, Altera, 1988). Od CPLD byl už pak jen malý krok k prvním FPGA obvodům (Xilinx, 1984). Dnes dostupná FPGA se ovšem od architektur z poloviny osmdesátých let významně odlišují. Trendem je pozvolný příklon k hrubozrnným architekturám; obvodům, které kromě elementárních programovatelných logických bloků obsahují i další komplexní podpůrné bloky. [8]

3.2.2 Základní vlastnosti FPGA

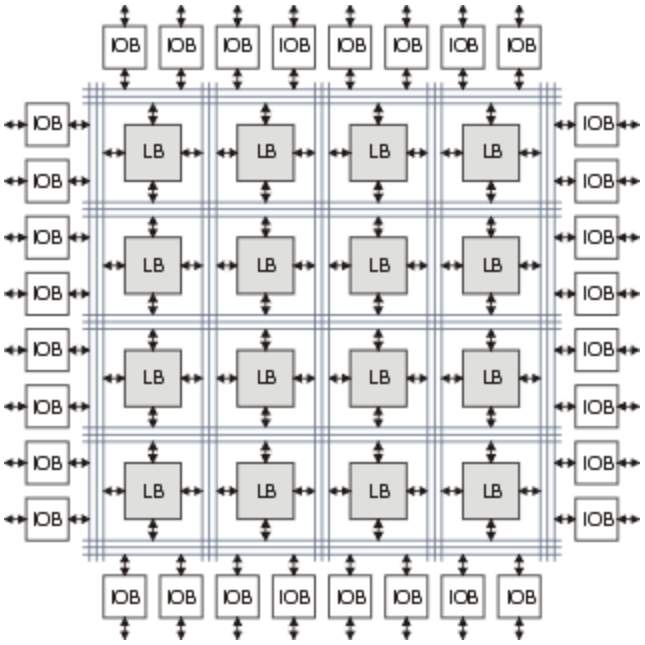
Základem programovatelného hradlového pole je logická buňka a volitelným propojením několika těchto buněk lze dosáhnout rozsáhlých komplexních funkcí, k jejichž realizaci by jinak bylo nutné použít mnoho různých obvodů. Tyto integrované obvody lze po vyrobení naprogramovat, tzn., že při použití technologie FPGA není třeba vyvíjet vlastní hardware tradičním způsobem, neboť tento lze definovat pomocí softwaru. Nespornou výhodou je, že bez specializovaného hardwaru má systém mnohem nižší výrobní náklady a navíc jej lze jednoduše upravovat a modernizovat pouhým přeprogramováním FPGA místo nákladné přestavby hardwaru. Díky tomu se lze soustředit na jiné činnosti, například na vývoj DSP a měřicích algoritmů, či na implementaci digitálních protokolů.

Souhrnně lze tedy říci, že k hlavním přínosům FPGA patří flexibilita - FPGA je rychle přeprogramovatelné pro nové funkce nebo vyšší výkon, dále je možno stejný hardware použít pro mnoho zákazníků a zanedbatelné nejsou samozřejmě ani nižší pořizovací náklady.

3.2.3 Architektura FPGA

Jak už bylo zmíněno výše, FPGA mají z programovatelných obvodů nejobecnější strukturu a obsahují nejvíce logiky. Jejich architektura je určena výběrem technologie programování, strukturou logických bloků, jejich rozmístěním a strukturou propojovacích vodičů.

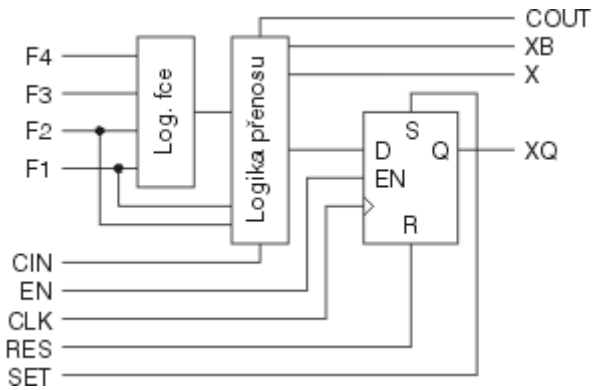
Základním prvkem obvodů FPGA jsou poměrně malé logické buňky, obvykle se čtyřmi až šesti vstupy, z nichž jsou logické funkce vytvářeny strukturou zvanou LUT – Look-up Table, připomínající malou paměť PROM. Pro vytvoření funkcí s větším počtem vstupů se tyto buňky musí spojovat, vytvářejí se bloky buněk a typické FPGA je vlastně kombinací univerzálních logických bloků a propojovacích vodičů. Jeho strukturu znázorňuje ***obrázek 3.1***.



Obrázek 3.1: *Struktura obvodu FPGA* [6]

Bloky LB (Logic Block) jsou vlastní programovatelné logické bloky, tvořené funkčními jednotkami, které vykonávají funkci zadanou návrhářem a konfiguračními prvky (multiplexory), jež dohromady spojují funkční jednotky podle informace o konfiguraci nahrané do FPGA. Jejich volitelným propojením lze dosáhnout rozsáhlých komplexních funkcí, k jejichž realizaci by jinak bylo nutné použít mnoho různých obvodů.

Bloky IOB (Input/Output Block) jsou vstupně-výstupní obvody pro každý pin FPGA, které obklopují pole bloků LB. Obvykle také obsahují klopné obvody, nikoli však kombinační logiku. Jsou připojeny k vývodům obvodu FPGA a jejich úkolem je propojení vnějších signálů se signály v poli bloků LB. Obsahují zpravidla registr, budič, multiplexor a ochranné obvody.



Obrázek 3.2*: Struktura konfigurovatelného logického bloku*

Kromě těchto bloků, které realizují základní logické kombinační a sekvenční funkce, vkládají výrobci do FPGA další specializované prvky, jako např. RAM (random-access memory) pro ukládání většího množství dat, PLL (Phase Locked Loop) nebo DLL (Delay Locked Loop) pro obnovení charakteristik hodinového signálu, případně pro násobení nebo dělení jeho frekvence, hardwarové násobičky, celé bloky pro aritmetické operace, rozhraní pro sériovou komunikaci s rychlostí v řádu gigabitů za sekundu, celá procesorová jádra a dokonce i konfigurovatelné analogové bloky, AD převodníky či paměť typu flash.

Všechny bloky mohou být různě propojeny pomocí globální propojovací matice, ale FPGA umožňují propojit některé signály sousedících bloků přímo, bez použití této matice. Toho se využívá např. u sčítaček nebo násobiček při realizaci rychlých obvodů šíření přenosu, neboť tyto přímé spoje mají mnohem menší zpoždění. Propojovací vodiče mají u všech vyráběných obvodů v podstatě stejný charakter. Tvoří je několik vrstev, připomínajících kovové vrstvy klasických ASIC, které se liší rozpětím základního prvku vodiče.

Technika programování používaná v FPGA je založena na programovatelném přepínači, který je obvykle tvořen buď tzv. **antipojistkou** (antifuse), paměťovou **buňkou SRAM** nebo **buňkou FLASH** a podle způsobu propojení tedy rozlišujeme hradlová pole těchto tří typů.

Antifuse – jedná se o prosté propojení signálových vodičů. Spínacím elementem je antipojistka (spoj se elektrickým signálem vytváří na rozdíl od pojistky, která spoj přerušuje). Antipojistka je vytvořena křížením dvou kovových vrstev vyplněných amorfním křemíkem. Při zvýšení napětí procházejícího křížením, změní křemík svoji strukturu, stane se vodivýma a dojde ke spojení kovových vrstev. Z toho je zřejmé, že antipojistku lze naprogramovat jen jednou, což ji znevýhodňuje oproti jiným ostatním technologiím. Vyznačuje se ale malým odporem přechodu, malými rozměry a úplnou testovatelností při výrobě nebo v programátoru. Její nespornou výhodou je okamžitá činnost po připojení napájecího napětí a taky skutečnost, že výsledný obvod neobsahuje paměť se souborem konfiguračních bitů (configuration bitstream), což zabraňuje pořizování pirátských kopií.

Programovací buňka SRAM má mnohem větší rozměry než antipojistka, skládá ze šesti tranzistorů, tvořících statickou paměťovou buňku, čímž zabírá na čipu mnohem více místa. Další nevýhodou je to, že při ztrátě napájecího napětí dojde ke ztrátě konfigurace a proto musejí být tato FPGA doplněna trvalou, na dodávce energie nezávislou pamětí obsahující soubor konfiguračních bitů, který se použije při zapnutí obvodu nebo při obnovování jeho činnosti po výpadku energie. To, že data musí být vždy znovu nahrána z vnější paměti, může vést k snadnějšímu pořizování nelegálních kopií. Její největší předností je to, že na rozdíl od předchozí technologie se pole tohoto typu dají přeprogramovat, v některých případech dokonce za chodu. Funkci buňky lze plně ověřit při výrobě FPGA.

Technologie FLASH – dalo by se říct, že tato technologie je kombinací obou výše uvedených. Umožňuje jak programování v aplikaci, tak před vlastním použitím. Konfigurační data, která

jsou reprogramovatelná, se permanentně ukládají v čipu a není tedy nutno připojovat externí paměť. Spotřeba energie je zhruba mezi výše uvedenými dvěma obvody.

3.2.4 Uplatnění hradlových polí FPGA

Ze všeho výše popsaného vyplývá, že použití FPGA je výhodné:

- Pro aplikace, které vyžadují zpracování velkých datových toků, pro výpočetně náročné algoritmy (multimédia – audio/video, 3G sítě, obecné číslicové zpracování signálu, kryptografie), kde nestačí klasické procesorové řešení svým výkonem a plánujeme příliš malou sérii, aby se vyplatilo navrhnout rovnou ASIC.
- Pro implementaci algoritmů či protokolů, které nejsou ještě normalizovány. Programovatelnost v poli i v cílové aplikaci umožní později přeprogramovat jednou vyrobené systémy, a tak přizpůsobit už prodaná zařízení novým požadavkům. Použití FPGA se zde vyplatí i pro velké série, protože se finální aplikace dostane na trh dříve. To přináší významnou konkurenční výhodu. Později lze z FPGA udělat ASIC a vyrábět dále zařízení s nižší cenou, avšak fixní funkcí (např. po standardizaci protokolu).
- Všude tam, kde je třeba speciální hardware, ale kvůli velikosti série se nevyplatí ASIC.
- FPGA se může hodit i během návrhu zákaznického integrovaného obvodu, kdy může sloužit jako platforma pro rychlé prototypování. Realizujeme-li například systém, který obsahuje několik procesorů, jež je nutno naprogramovat (spotřební elektronika – multimédia, herní konzoly), může nám FPGA prototyp významně zkrátit dobu návrhu. Software totiž může být vyladěn už během implementace vlastního návrhu ASIC – tzv. software-hardware codesign.
- Programovatelná hradlová pole najdou brzy aplikaci i v osobních počítačích ve formě rekonfigurovatelného akcelerátoru pro obecné výpočty. [7]

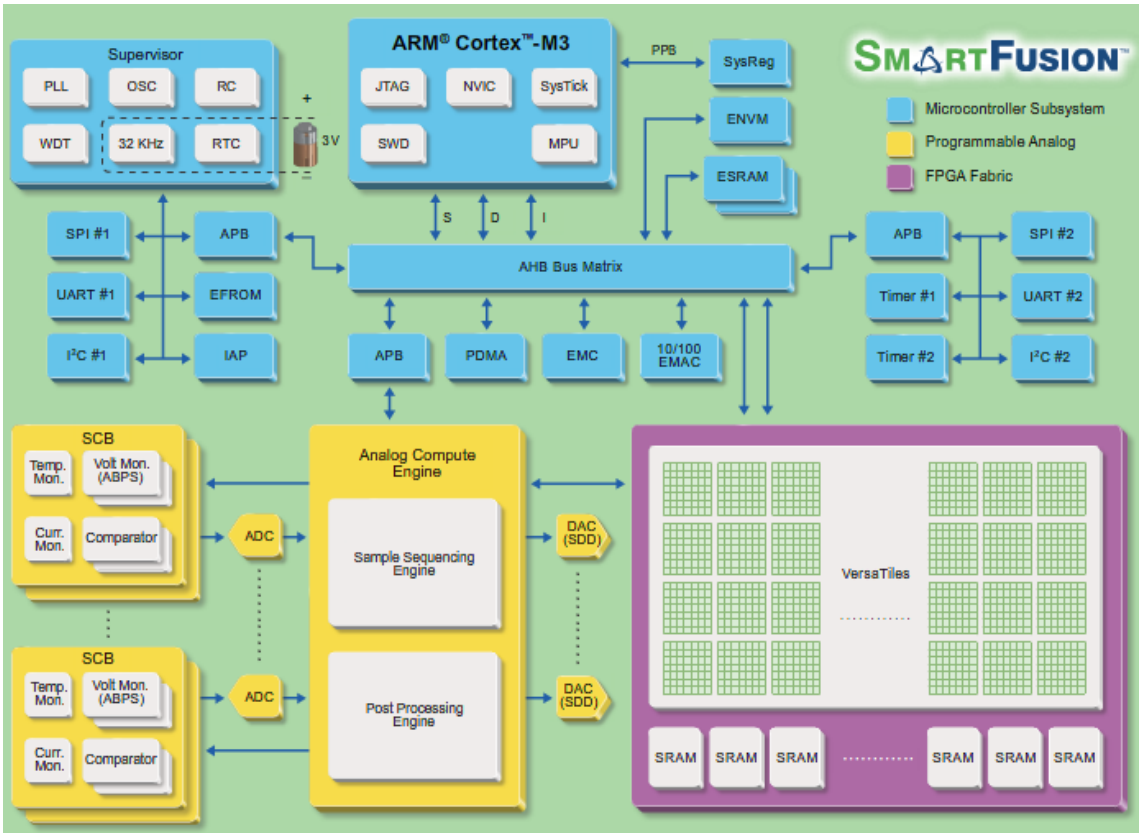
V současnosti FPGA vyrábějí zejména firmy Actel, Altera, Atmel, Lattice a Xilinx a pro svoji flexibilitu nacházejí stále větší uplatnění v různých oborech lidské činnosti jako v telekomunikační technice, v obrazových přenosových trasách, v automobilovém průmyslu, v lékařství i jinde.

4. Programovatelné hradlové pole rodiny SmartFusion

K mé práci bylo využito programovatelné hradlové pole rodiny SmartFusion, produkované společností Actel. V této kapitole si proto povíme o tomto hradlovém poli něco blíže.

SmartFusion je zařízení, které v sobě současně zahrnuje, kromě samotného FPGA, také mikrokontrolerový subsystém (MSS) s procesorem ARM Cortex-M3 a z hlediska šíření signálu velmi rychlou programovatelnou analogovou část. Název SmartFusion, který by se dal česky přeložit také jako "elegantní splynutí" vznikl právě z integrace všech tří částí do jediného čipu, což má za následek nejen to, že čip zabírá velmi malou plochu, má nízkou spotřebu energie, ale také to, že má mnohem nižší pořizovací cenu.

Podrobnější strukturu a části SmartFusion zařízení můžeme vidět na následujícím obrázku.



***Obrázek 4.1**: Schéma SmartFusion zařízení* [10]

V tomto schématu jsou modře označeny části, které pokrývají mikrokontrolerový subsystém, žlutě pak části analogové struktury a nakonec zde můžeme vidět samotné programovatelné hradlové pole, znázorněné fialovou barvou.

Zařízení SmartFusion jsou podporována potřebným vývojovým systémem Libero, rovněž poskytovaným společností Actel, který bylo pro realizaci práce nutno použít a o kterém se budu ještě zmiňovat v průběhu práce.

Firma Actel nabízí hned několik druhů zařízení této rodiny, které se liší v některých parametrech, jako například počtem hradel programovatelného pole, velikostí Flash paměti, počtem periferních zařízení atd., avšak koncept tří integrovaných technologii na jednom čipu je u všech těchto SmartFusion obvodů stejný. K mé práci byl vybrán obvod A2F200, který se nachází na mnou použité vývojové desce, označené A2F-EVAL-KIT.

Podrobnější informace o SmartFusion technologii a jejich částech se můžete dočíst v dokumentaci na stránkách společnosti Actel. [10]

4.1 MSS

MSS se skládá ze 100 MHz Cortex™-M3 procesoru a integrovaných periferních zařízení, které jsou s ním propojeny přes vícevrstvou sběrnici AHB bus Matrix (ABM). MSS pro připojení do programovatelného hradlového pole využívá speciální konfigurovatelný blok FIC (Fabric Interface Controller), který umožňuje přemostění mezi sběrnicemi. O sběrnicích a jejich přemostěních budou podrobněji pojednávat podkapitoly Hardwarového návrhu. K integrovaným perifériím, kromě bloku FIC, patří i například SPI, I2C, UART sériový port, vestavěná FlashROM (EFROM), 10/100 Ethernet MAC, časovače, oscilátory a další.

K MSS části SmartFusion zařízení se ještě vrátím v kapitole 7.3, kde bude popsána konfigurace MSS, která je nedílnou součástí tvorby mého systému.

4.2 Analogová část

Programovatelná analogová část se skládá z bloků SCB (Signal Conditioning Block), které mohou zpracovávat a obsluhovat více signálů najednou. Tyto bloky upravují a optimalizují analogový signál pro následné AD převodníky, které jsou rovněž součástí tohoto segmentu. Dalšími součástmi jsou i DA převodníky a především pak analogová výpočetní jednotka ACE (Analog Compute Engine). Hlavním úkolem této jednotky je vypomoci procesoru Cortex-M3 a odlehčit mu práci od zpracování analogových bloků. Ve srovnání se systémy, kde má hlavní procesor na starosti i řízení analogové části, by pomoci tohoto řešení měla být poskytnuta lepší propustnost systému a jeho spotřeba energie.

4.3 Programovatelné hradlové pole

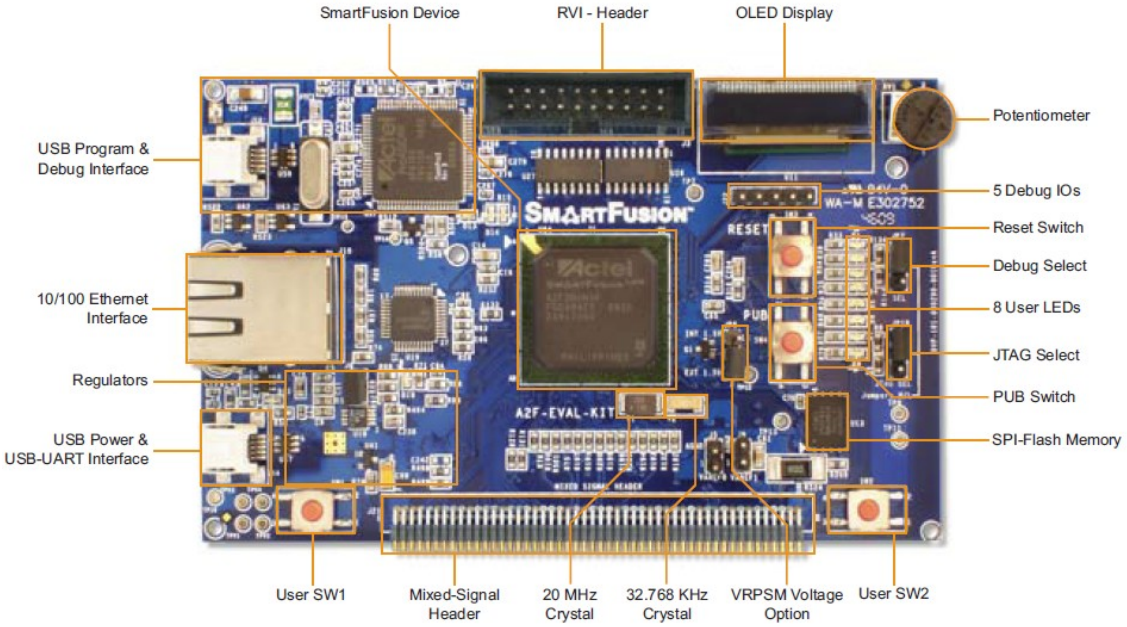
Programovatelné hradlové pole obsažené v SmartFusion zařízení je postaveno na osvědčené nízkoenergetické Flash FPGA architektuře ProASIC3, která je dostatečně odolná vůči chybám.

Jednou z hlavních předností tohoto hradlového pole integrovaného v SmartFusion zařízení je právě to, že je postaveno na Flash architektuře. Ta zajistí, že pole je rekonfigurovatelné a stav konfigurace v něm zůstává stejný i po odpojení od zdroje elektrické energie. Tím pádem při odpojení a znovu přivedení elektrického napětí nepotřebuje k jeho nastavení pokaždé načítat data z externí zaváděcí paměti, jako je to například u SRAM polí. Bezpečnostní mechanizmy na desce pak zabraňují v neoprávněném přístupu k programovacím informacím a umožňují návrháři bezpečnou vzdálenou aktualizaci FPGA obvodu. Tyto mechanizmy jsou zajištěny pomocí standardu AES (Advanced Encryption Standard), což je standard pro šifrování pomocí symetrické blokové šifry.

Výhody tohoto pole pro návrháře jsou kromě výše zmíněného také především nízká cena, velký výkon a snadnost použití.

4.4 SmartFusion Evaluation Kit

SmartFusion Evaluation Kit je také označení pro desku A2F-EVAL-KIT. Centrální komponentou této desky je samotné SmartFusion zařízení, které pro komunikaci s okolím využívá dalších komponent a rozhraní, které jsou vestavěny na desce.



***Obrázek 4.2**: SmartFusion Evaluation Kit (A2F-EVAL-KIT)*[11]

Deska SmartFusion Evaluation Kit obsahuje:

- SmartFusion zařízení A2F200M3F-FGG484, které obsahuje:
  - 200 000 hradel, 256 KB FLASH paměť, 64 KB SRAM, přídavnou distribuovanou SRAM paměť v FPGA poli, řadič pro externí paměť,
  - Mikrokontroler subsystém s procesorem ARM Cortex-M3 a periferními zařízeními Ethernet, DMA, I2C, UART, časovače, AD a DA převodníky a další
- SPI-flash paměť (8MB)
- USB rozhraní pro programování a odlaďování navrhovaného systému
- USB UART rozhraní pro sériovou komunikaci (toto rozhraní slouží také pro přívod napájení desky)
- 10/100 Ethernet rozhraní
- RVI rozhraní pro programování a laděni aplikací ze systému Keil nebo IAR
- Mixed-signal rozhraní pro připojení dceřiné desky

Z uživatelských vstupů jsou to pak komponenty:

- OLED display (96×16 pixel) s I2C rozhraním
- Potenciometr, k regulování a monitorování napětí
- 8 uživatelských LED diod
- 2 uživatelské tlačítka
- Monitorování teploty pomocí teplotní diody a další

Více informací o desce A2F-EVAL-KIT se můžeme dočíst v její dokumentaci na stránkách firmy Actel. [11]

5. Jazyk VHDL

Moderní FPGA jsou velmi komplexní zařízení s miliony ekvivalentních hradel. Návrh obvodů, které jsou na FPGA založeny, se tudíž neobejde bez specializovaných vývojových nástrojů. Jedním z nich je programovací jazyk pro popis hardwaru, jehož textový popis návrhu se převede na netlist, využívající obecné logické bloky. Netlist je seznam logických bloků, vstupních a výstupních portů a všech spojů vyvíjeného obvodu, který definuje výsledný logický obvod. V dnešní době se pro popis hardwarových struktur hradlových polí používají dva nejrozšířenější, co se funkcí týče rovnocenné, jazyky a to Verilog , rozšířený především v Americe a Asii a jazyk VHDL, používaný především v Evropě a tedy i u nás.

Základy jazyka VHDL byly položeny v roce 1980, kdy vznikl jako „vedlejší produkt" výzkumného projektu rychlých integrovaných obvodů, financovaného ministerstvem obrany USA. Od r. 1987 je standardem IEEE (Institute of Electrical and Electronics Engineers), v roce 1997 byl revidován a je použitelný i pro návrh analogových obvodů. Jedná se o typovaný programovací jazyk, který má prostředky pro popis paralelismu, konektivity a explicitní vyjádření času. Jazyk VHDL se používá jak pro simulaci obvodů, tak i pro popis integrovaných obvodů, které se mají vyrábět. Zkratka VHDL znamená VHSIC Hardware Description Language (česky jazyk pro popis hardware), kde VHSIC je zkratka z Very-High-Speed Integrated Circuit (česky velmi rychlé integrované obvody). [9]

- VHDL může být použitý pro popis, ověření a syntézu velmi rozsáhlých digitálních projektů, což je jeden z jeho klíčových rysů, protože jeden a ten samý VHDL kód plní všechny tyto tři účely.
- Umožňuje návrh jak logických tak i sekvenčních struktur a jeho hlavní výhodou je jeho univerzálnost.
- Pomocí jazyka VHDL získáme technologicky nezávislý popis logického obvodu na relativně vysoké úrovni abstrakce. Díky tomu je možné jeden a ten samý popis obvodu snadno nezávisle přenášet a využívat mezi různými architekturami FPGA. Na návrhu tedy můžeme začít pracovat bez znalosti cílového obvodu a konečná implementace navržené struktury je závislá až na kompilaci VHDL kódu, tzn., že pomocí tohoto jazyka lze provádět návrhy pro hradlová pole většiny výrobců a následně použít vhodný kompilátor, který lze volně stáhnout z internetových stránek jednotlivých výrobců.
- Jedná se o otevřený standard, tzn., že na rozdíl od jiných programovacích jazyků nepotřebujeme k jeho použití licenci vlastníka.
- Jedním z dalších rozdílů oproti běžným programovacím jazykům je mimo jiné i fakt, že jednotlivé příkazy se vykonávají paralelně. V případě, že potřebujeme vykonávat příkazy sekvenčně za sebou, je možné je vložit do speciálních bloků ohraničených klíčovým slovem *process*. Mimo tyto bloky se příkazy vykonávají paralelně.

- Při práci s tímto jazykem je návrh struktury možné a dokonce i vhodné rozdělit na samostatné bloky, které jsou následně spojeny ve finálním modulu pomocí objektu typu signál, který většinou odpovídá reálnému elektrickému signálu.

Jazyk VHDL má dvě povinné komponenty, které jsou nezbytné v každém modulu. Jedná se o:

- **Komponentu Entity** - zprostředkovává vnější propojení pomocí vstupních a výstupních portů. Pokud je modul pouze součástí rozsáhlejšího návrhu, slouží jeho výstupní signály k propojení s nadřazeným modulem. Pokud je tento modul nejvyšším v hierarchii, jsou jeho porty fyzicky připojeny na vývody hradlového pole.
- **Komponentu Architecture** - tato povinná komponenta popisuje chování Entity (vnitřní strukturu obvodu). Má dvě části: deklarační, která slouží pro deklaraci signálů, proměnných apod. a část příkazovou s popisem vlastní logické funkce obvodu.

6. Systémový návrh

Prvním krokem při vytváření jakéhokoliv systému by vždy měla být specifikace požadavků a ujasnění celkového, ale i detailnějšího pohledu a způsobu tvorby tohoto budoucího systému. Proto i já se v této kapitole pokusím sepsat a rozebrat to, co se od tvořeného systému vyžaduje a jak by měl správně fungovat.

Požadovaný systém by měl zajišťovat měření jevu zvaného akustická emise, a tedy zachycovat a vyhodnocovat vady v konstrukci a materiálu snímaného zařízení, pomocí nedestruktivní akustické metody měření.

Systém by měl být vhodný pro řešení úloh v oblasti kontroly výrobní technologie a zjišťování technického stavu ocelových a železných konstrukčních částí. Zjištěné údaje pak mohou být podkladem například pro úpravy technologických režimů a pro odhad čerpání životnosti konstrukcí.

Požadovaný systém by měl rovněž umožňovat trvalou diagnostiku za provozu a tedy snímat emitovaný signál na pozorovaném zařízení v jeho provozních podmínkách. To umožní kontrolovat kvalitu provedených svarů, průběh tepelného zpracování výkovků a tuhnutí velkých odlitků. Především z tohoto důvodu pak bude celé měřicí zařízení umístěno v ochranné uzavřené hliníkové skříni s krytím IP 65 (**IP 6x** - Zařízení je prachotěsné a je chráněno před dotykem drátem, **IP x5** – zařízení je chráněno proti tryskající vodě). Tímto by měl být zajištěn chod a ochrana měřícího systému během provozních podmínek snímaného zařízení.

Jak nám již mohla přiblížit kapitola 2 pojednávající o akustické emisi, měření akustické emise je samozřejmě složitý proces, který v sobě zahrnuje mnoho dílčích, postupně navazujících měřicích a vyhodnocovacích postupů a části, z kterých se pak postupně může sestavit celý robustní, komplexní, měřicí systém. Dílčí části systému by pak měly pokrývat především tyto činnosti:

- Registrace a lokalizace zdrojů emisních událostí
- Vyhodnocování jejich závažností a možností dalšího šíření.

Především z důvodů rozsáhlosti celého řešení se má práce bude zabývat pouze první z uvedených činností, a to sběrem a zpracováním dat pro lokalizaci zdrojů emisních událostí na snímaném zařízení.

6.1 Systémové požadavky

Činnost lokalizace zdrojů emisních událostí postupuje od naměření údajů o příchodu signálu vzruchu prostřednictvím snímačů, přes jejich hardwarové zpracování následováno softwarovým odečítáním a zpracováním těchto údajů, až po jejich přeposlání nadřazenému systému přes ethernet rozhranní, za účelem dalšího vyhodnocování.

Místo zdroje vzniku emisní události je možné určit dvěma postupy:

1. Z amplitud RMS hodnot signálu při známém útlumu šíření.
2. Měřením časových rozdílů příchodu signálu na snímače.

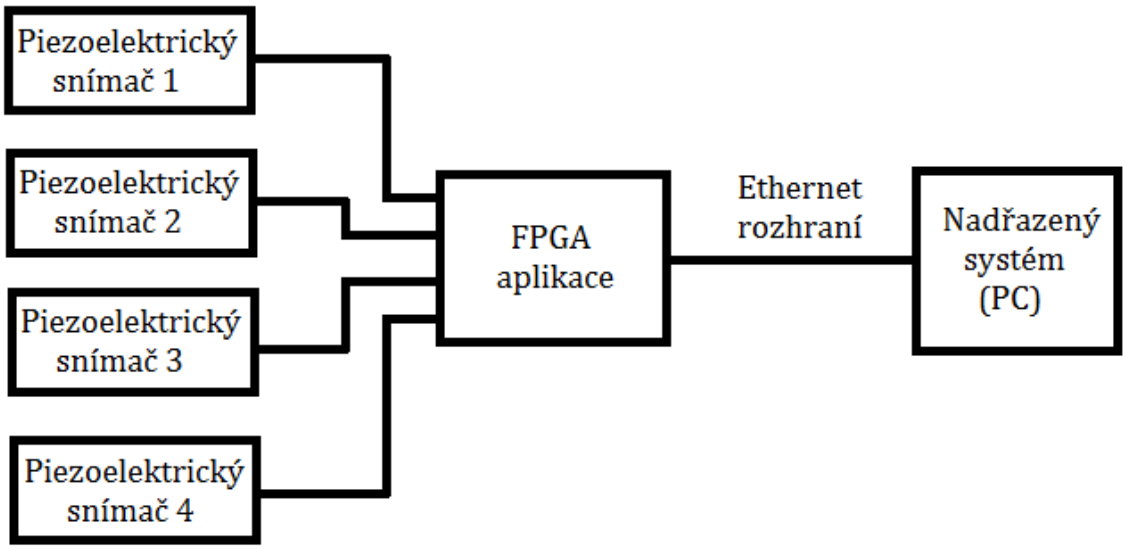
Já si při tvorbě svého systému zvolil druhý z postupů. Pro získávání základních informací o poloze epicentra vzruchu bude tedy třeba měřit dobu zpoždění příchodu signálu vzruchu mezi jednotlivými piezoelektrickými snímači vzruchu. K dispozici bude mít systém čtyři piezoelektrické snímače, které budou rozmístěny na ocelové či kovové konstrukci, kde se signál zvuku šíří přibližně stejnou rychlostí. Jejich velikost by neměla překročit 30m. Měření by mělo probíhat s přesností minimálně na 1mm. Výsledky měření se budou softwarově zpracovávat a posílat přes ethernet linku nadřazenému systému k dalšímu vyhodnocování.

V bodech je pak souhrn požadavků na mnou realizovaný subsystém následující:

- Systém bude sbírat a zpracovávat data pro lokalizaci zdrojů emisních událostí na snímaném zařízení a to prostřednictvím piezoelektrických snímačů.
- Kontrolované bloky budou ocelové či kovové konstrukční části o velikosti nepřekračující 30m (ve všech třech souřadnicových osách).
- Emisní signály vzruchu budou zpracovávány ve speciálním obvodu, který je schopen snímat a zpracovávat signály ze čtyř snímačů.
- Určení polohy epicentra vzniku emisní události by mělo probíhat s přesností minimálně 1mm.
- Činnost samotného systému bude nezávislá na řídicím nadřazeném počítači, nicméně mu bude posílat data pro další zpracování a vyhodnocování.
- Komunikace s nadřazeným PC bude probíhat přes ethernet rozhranní.
- Systém bude realizován na vývojové desce plošných spojů A2F-EVAL-KIT od společnosti Actel, která obsahuje kromě FPGA pole rodiny SmartFusion a mikrokontroleru subsystému také mnoho dalších potřebných periférií.

Celý mnou vytvářený subsystém bude rozdělen na dvě části, hardwarovou a softwarovou. Hardwarová část bude řešit samotné měření časových rozdílů příchodu signálu na snímače. Softwarová část pak bude řešit čtení dat ze sběrnice z hardwarové části i jejich další zpracování a přeposlání k další analýze do nadřazeného systému přes ethernet linku. Hardwarový návrh bude proveden pomocí programovacího jazyka VHDL a vývojového prostředí Libero IDE na hradlovém poli rodiny SmartFusion, umístěném na vývojovém kitu A2F-EVAL-KIT poskytovaném společností Actel. Softwarový návrh pak bude uskutečněn ve vývojovém prostředí SoftConsole v programovacím jazyku C.

Blokové schéma navrhovaného systému pak můžeme vidět na ***obrázku 6.1.***



Obrázek 6.1*: Blokové schéma navrhovaného systému*

7. Hardwarový návrh

Jak už bylo zmíněno v kapitole výše, aby měl nadřazený systém informace pro další vyhodnocování polohy epicentra vzruchu, je třeba nějakým způsobem měřit dobu zpoždění příchodu signálu vzruchu mezi jednotlivými piezoelektrickými snímači vzruchu. Čas, který urazí signál vzruchu od svého prvního kontaktu s některým ze snímačů k dalším snímačům, bude měřen hardwarovými čítači, čítajícími zvolenou frekvencí, odpovídající jistým parametrům pro šíření zvuku v kovových materiálech, o kterých se ještě zmíníme v jedné z následujících podkapitol. Kromě těchto čítačů, jakožto hlavních stavebních prvků, se k správné logice čítání musí do obvodu tvořeného v hradlovém poli přidat ještě další logické bloky, umožňující správnou činnost čítačů a vyhodnocení ukončení měření. Celá měřící logika pak musí spolupracovat se sběrnicí pro správné připojení výstupů čítačů k procesoru. K tomuto účelu bylo využito jádro CoreAPB3 sběrnice APB.

Pro komunikaci s nadřazeným systémem bylo dle pokynů zadání práce využito Ethernet rozhraní. Pro snazší ověřování a vizualizaci dat na počítači jsem ale zprvu využil sériové rozhraní UART. Rozhraní Ethernet i UART jsou přímo vestavěné ve SmartFusion MSS. Z jádra mikrokontroléru SmartFusion MSS byly kromě těchto dvou rozhranní využity i další důležité části jako například Clock Management, Reset Management, Fabric Interface, o jejichž nastavení se zmíním ještě později. Funkčnost hardwarového návrhu měřící logiky provedené na programovatelném hradlovém poli si na závěr ověřím pomocí simulace, provedené v nástroji ModelSim, který je zahrnutý v prostředí Libero IDE. Celou výše uvedenou problematiku podrobněji rozeberu a popíšu v následujících kapitolách.

7.1 Čítače a jejich návrh

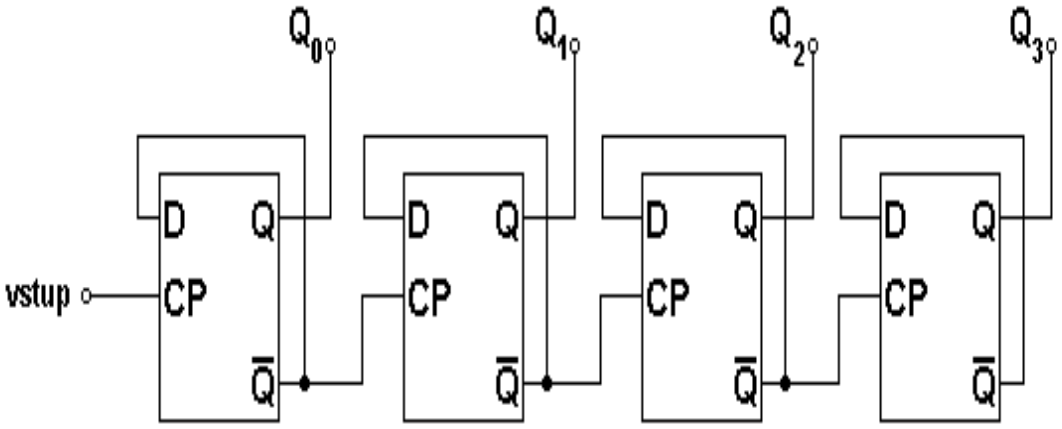
Před samotným popisem návrhu měřícího obvodu a čítačů, které jsou jeho hlavní součástí, si o tomto logickém segmentu povíme něco blíže.

7.1.1 Čítače

Čítač je sekvenční logický obvod, který čítá impulsy v závislosti na sledovaném signálu, kterým nejčastěji bývá hodinový signál. Spravuje určité paměťové místo, ke kterému tedy přičítá jedničku na základě zjištění náběžné nebo sestupné hrany sledovaného signálu. Tento obvod se skládá z několika klopných obvodů, přičemž počet klopných obvodů nám udává kolika bitový čítač je. Čítače pracují buď jako synchronní, kdy mají všechny klopné obvody společný hodinový signál, nebo jako asynchronní, kdy je hodinový signál přiveden pouze na vstup prvního klopného obvodu a do zbylých klopných obvodů je na vstup hodinového signálu přiveden pouze výstup předchozího klopného obvodu. Pro čítače se používají buď klopné obvody typu D, které mohou reagovat buď na náběžnou nebo na sestupnou hranu hodinového signálu, nebo klopné obvody typu JK, které reagují však pouze na sestupnou hranu. Čítače

mohou být také obousměrné, tedy schopny čítat jak vpřed (inkrementace hodnoty), tak i vzad (dekrementace hodnoty).

Pro představu můžeme na obrázku ***obrázeku 7.1*** vidět 4 bitový asynchronní čítač sestavený z D klopných obvodů.



Obrázek 7.1: *Ukázka sestavení asynchronního čítače s KO typu D*

Obecný čítač často mívá kromě sledovaného vstupního signálu (označovaného většinou jako CLK – hodinový signál) i několik dalších řídících vstupů, které rozšiřují jeho možnosti.

- RESET - vstup nulování čítače
- UP / DOWN - vstupy k určování směru čítání u obousměrných čítačů
- SET - pro nastavení hodnoty čítače na požadovanou nenulovou hodnotu
- ENABLE - povolovací vstup, kdy čítání probíhá pouze v případě, že je tento signál aktivní
- případně další signály

Na výstup se pak vyvádí signál určující číselnou hodnotu daného čítače, často spolu s dalšími informacemi jako např. přetečení či podtečení rozsahu čítání apod.

Jelikož já si hardware navrhuji za pomocí jazyka VHDL, nemusím se zabývat skládáním jednotlivých logických bloků klopných obvodů, ale stačí mi jen v tomto jazyce správně zpracovat a vyhodnotit příchozí vstupní signály a výsledky této logiky čítání přivést na výstupy. O vygenerování a správné sestavení logického obvodu se postarají příslušné nástroje v prostředí Libero IDE.

7.1.2 Návrh čítače

Jak bylo popsáno v kapitole 6, potřebujeme zajistit to, aby po prvním příchodu vzruchu na některý z čítačů tento čítač nečítal a spustilo se čítání všech ostatních čítačů. Po postupném příchodu vzruchu na další čítače se pak na příslušných komponentách čítání zastaví. K řešení tohoto problému se musely jako vstupní signály přivést kromě nulovacího vstupu RESET a CLK vstupu určujícího jeho požadovanou čítací frekvenci, také dva vstupy ENABLE. Na základě správné kombinace těchto dvou signálů je pak čítač připraven čítat či nečítat. Vstupy jsme si označili EN1, EN2.

- Signál EN1 nám říká, zda vzruch dorazil na snímač spjatý s tímto příslušným čítačem, a zakazuje čítání tomuto konkrétnímu čítači.
- Vstup EN2 nám pak udává informaci o tom, zda alespoň na jeden snímač dorazil signál vzruchu a povoluje čítání všem čítačům.

Při výše zmíněném požadovaném chování je pak jasné že čítač musí počítat, tedy inkrementovat svou hodnotu při kombinaci vstupů:

- EN1=0, tedy na čítač ještě nedorazil vzruch,
- EN2=1, tedy že na alespoň jeden další čítač dorazil vzruch.

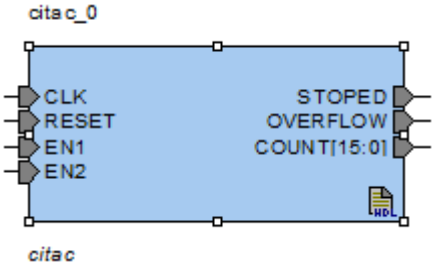
Výstupem čítačů jsou pak signály COUNT určující hodnotu čítače, dále pak signál OVERFLOW pro případ přetečení rozsahu čítače a STOPED určující stav ukončení čítání.

Část VHDL kódu zachycující logiku čítání čítače můžeme vidět na ***obrázku 7.2***.

```
...
if (RESET='0') then
   cnt := (others => '0');
elsif rising_edge(CLK) then
    if (EN1='0' AND EN2='1' AND cnt_overflow='0') then
            cnt := cnt + 1;
    end if;
end if;
...
```

***Obrázek 7.2**: Část VHDL kódu čítače*

Blok čítače má potom následující podobu.



Obrázek 7.3: *Blok navrženého čítače*

Z důvodů požadované kapacity čítače jsme zvolili 16 bitový výstup. Důvod této volby bude upřesněn v kapitole 7.4. V jazyku VHDL je tento výstupní signál reprezentován 16 bitovým vektorem typu `std_logic_vector`, který poskytuje standardní VHDL knihovna `IEEE.std_logic_1164.all`. Definice signálu pak vypadá takto:

```
COUNT: out std_logic_vector(15 downto 0);
```

Pro tento typ proměnné ovšem není definována operace sčítání. Nejedná se totiž o obyčejná čísla. Není tedy možné jednoduše k tomuto signálu přičítat jedničku přes operátor +. Řešením je buď napsání vlastní funkce, což by nejspíš nebylo tak obtížné, ale výhodnější je využít již napsané funkce v příslušných knihovnách. Vytvořil jsem si proto pomocnou proměnnou `cnt`, která je typu UNSIGNED. Tento typ je definován v balíku `numeric_std` knihovny IEEE a nad tímto typem je již definována operace sčítání. Abych tuto proměnnou mohl přiřadit signálu COUNT bylo ji ještě nutno explicitně přetypovat na `std_logic_vector`.

```
variable cnt: unsigned(15 downto 0);

cnt := cnt + 1;

COUNT <= std_logic_vector(cnt);
```

7.2 Návrh měřicího obvodu

V této podkapitole popíši propojení navržených bloků čítačů s dalšími prvky obvodu pro zajištění správné funkčnosti měření vzdálenosti vzruchu od jednotlivých snímačů.

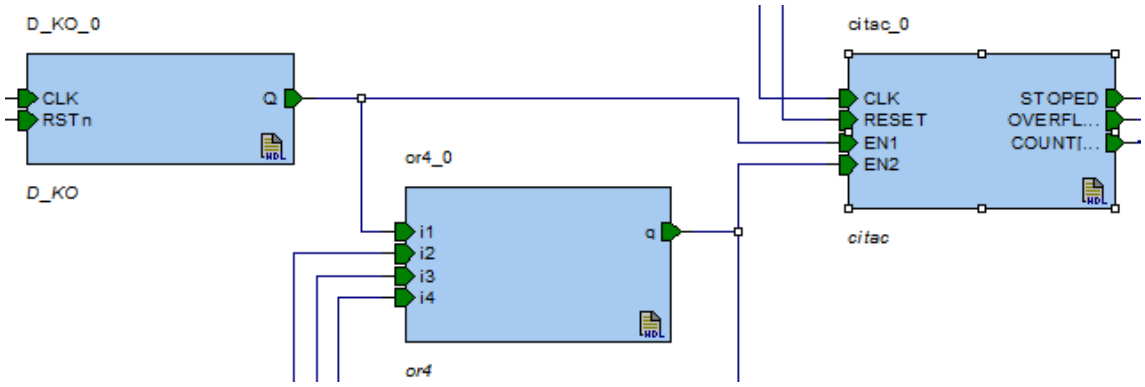
7.2.1 Zapojení vstupních signálů čítače

Aby bylo zajištěno, že na vstupy EN1a EN2 budou po celou dobu měření přiváděny správné informace, bylo nutné před čítač předřadit registrudávající, zda měřený signál vzruchu již dorazil na příslušný snímač. Tento registr, reprezentovaný klopným obvodem typu D (rovněž vytvořeným v jazyku VHDL), po obdržení první náběžné hrany sledovaného vzruchu na svůj

hodinový vstup pak nadále, dokud není resetován, na výstupu udržuje úroveň signálu na logické jedničce. Bez tohoto registru, umístěného mezi snímačem a čítačem, by se mohlo stát, že údaje na vstupech EN1 a EN2 by se přepínaly z logické nuly na logickou jedničku a naopak v závislosti na náběžné a sestupné hraně vzruchu a mohly by poskytnout nepřesná data. Výstupem tohoto registru je pak signál EN1.

Signál EN2 bude aktivní, pokud bude aktivní alespoň jeden ze signálů EN1. Získáme jej jednoduše pomocí čtyřvstupého hradla OR. Výstup z tohoto hradla pak přivedeme na EN2 vstupy čítačů.

Tímto mechanizmem jsme si také zajistili případ, kdy signál vzruchu dorazí na více snímačů ve stejný okamžik. Potom příslušné čítače zůstanou vypnuté a hodnoty budou načítat všechny ostatní.



***Obrázek 7.4**: Část schématu zapojení*

Výstupem celého měřícího obvodu budou pak jednotlivé napočítané hodnoty čítačů, informace o případném přetečení jednotlivých čítačů a údaj poskytující informaci pro nadřazený systém o tom, zda měření skončilo a můžou se začít vyhodnocovat naměřená data. Tento stav získáme v případě, že mají všechny čítače aktivní, tedy v logické jedničce, vývod STOPED.

Po této části vývoje systému, jsem přešel k ověření doposud navržené logiky, a to pomocí simulace. Této problematice se ale pro dodržení lepší souvislosti a komplexnosti popisu tvorby hardwarové části budu věnovat až později v podkapitolách 7.6.

7.2.2 Přivedení dat na sběrnici

Abychom mohli data, získaná z měřícího logického obvodu, procesorem dále zpracovávat pro nadřazený systém, bylo nutné tyto doposud realizované bloky samostatného logického obvodu připojit z venku na sběrnici procesorového jádra. To využívá sběrnicovou architekturu AMBA (Advanced Microprocessor Bus Architecture), která je specifikací definující komunikační standard pro návrh vestavěných mikrokontrolérů. V AMBA specifikaci je definováno několik

odlišných sběrnic. Kromě AHP (Advanced High-performance Bus) je zde definována také sběrnice APB (Advanced Peripheral Bus), kterou jsem využil při tvorbě svého návrhu. Tuto sběrnici si pro lepší pochopení a vysvětlení důvodu této volby stručně popíšeme v následující kapitole.

7.2.3 Obecná charakteristika APB sběrnice

Oproti typu sběrnice AHB, která je určena pro rychlou spolupráci mezi bloky jádra a rychlou komunikaci s pamětí, je APB sběrnice navržena pro komunikaci s pomalejšími periferními zařízeními, vystačí si s menším počtem hradel a je optimalizována pro nízkou spotřebu. Má také mnohem jednodušší protokol a tedy i počet komunikačních signálů v jejím rozhraní.

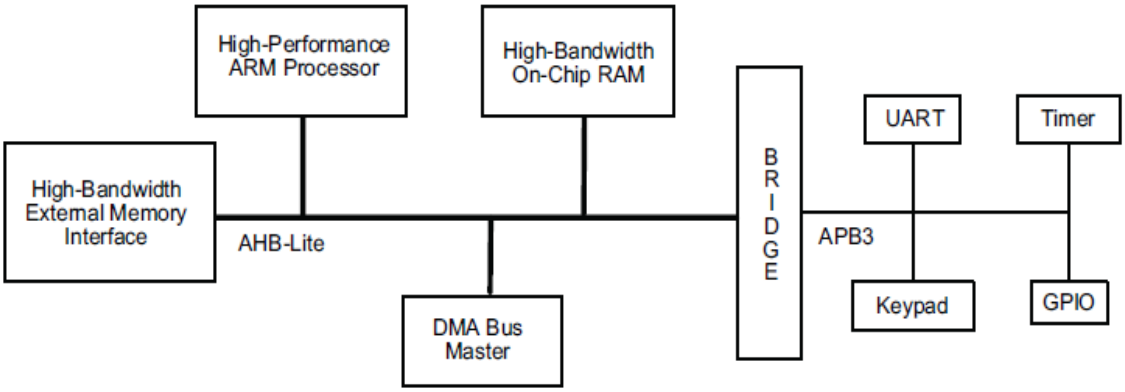
Z těchto důvodů pak uživatelem naprogramované aplikace pro FPGA, kterým v naprosté většině případů stačí relativně pomalejší přenos, komunikují s MSS pomocí mapování registrů přes sběrnici APB.

Já budu ve své práci pracovat se sběrnicí APB3 také známou jako APB V3, která je definována v AMBA 3 protokolu (třetí generace AMBA specifikace) a oproti APB umožňuje např. rozšíření přenosu nebo předávání informací o selhání.

Přehled charakteristických signálů poskytovaných sběrnicí APB3 se stručným popisem je uveden v následující tabulce.

- PCLK: Hodinový signál sběrnice. Všechny přenosy jsou synchronizovány dle tohoto signálu.
- PSEL: Signál unikátního výběru zařízení. Využívá se k výběru konkrétního zařízení z určité předdefinované škály adres.
- PADDR: Adresová část sběrnice. Používá se k adresování v rámci předem známého rozsahu adres. Může být až 32 bitová.
- PWRITE: Signál určující, zda se jedná o čtecí (PWRITE = Low) nebo zapisovací (PWRITE = High) cyklus
- PENABLE: Indikuje stupně přenosu dat.
- PREADY: Signál využívaný periferními zařízeními, kterým si může vynutit vložení čekacích cyklů sběrnice.
- PWDATA: datová část sběrnice pro zápis dat do periférií (může být až 32 bitová)
- PRDATA: datová část sběrnice pro čtení dat z periférií (může být až 32 bitová)
- PSLVERR: Používaná periferními zařízeními pro indikaci přenosové chyby.

Na následujícím ***obrázku 7.5*** je ještě pro ucelení zobrazen typický AMBA systém, kde můžeme vidět ARM procesor připojený k AHB-Lite sběrnici a přes můstek (bridge) k APB sběrnici. Vidíme také, že k AHB-Lite jsou připojené periférie a paměti pro rychlou komunikaci, kdežto UART, GPIO atd. spadají do periférií pomalých.



***Obrázek 7.5**: Typický AMBA systém* [12]

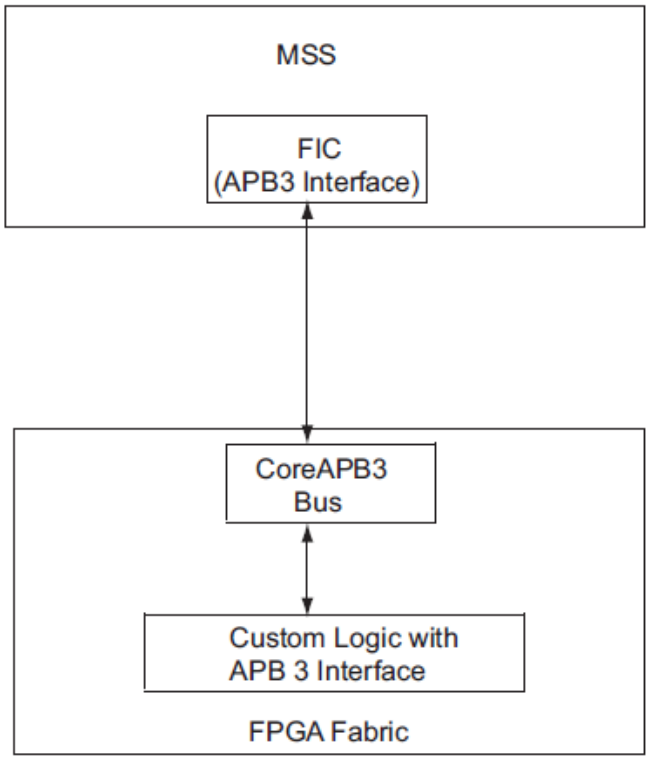
7.2.4 Můstky mezi MSS-APB-Periferií

K samotnému propojení periferních zařízení vytvořených v programovatelném hradlovém poli FPGA s MSS přes APB3 je zapotřebí dvou můstků. A to mezi samotným MSS a APB3 a mezi vlastním logickým obvodem a APB3.

MSS propojuje Procesor Cortex™-M3 a Integrované periférie přes více vrstvou AHB Bus Matrix (ABM). Pro připojení MSS do programovatelného hradlového pole se využívá konfigurovatelný blok FIC (Fabric interface controller), který umožňuje přemostění AHB - AHB nebo AHB - APB3 a to mezi ABM a AHB nebo APB3 sběrnicí, které uživatel realizoval v FPGA návrhu.

Druhým z můstků je rozhraní APB3 slave, které funguje jako most mezi APB3 sběrnicí a periferními zařízeními, které jsou ke sběrnici připojeny. To přijímá signály z APB3 a převádí je do podoby srozumitelné připojené periferii a totéž provádí i v opačném směru (signály z periférie převádí do podoby srozumitelné pro APB3).

Následující ***obrázek 7.6*** nám zobrazuje, jak je připojen vlastní uživatelský logický blok s APB3 rozhranním do samotného MSS přes CoreAPB3 v zařízení SmartFusion.



***Obrázek 7.6**: Schéma zapojení bloku hradlového pole do MSS* [12]

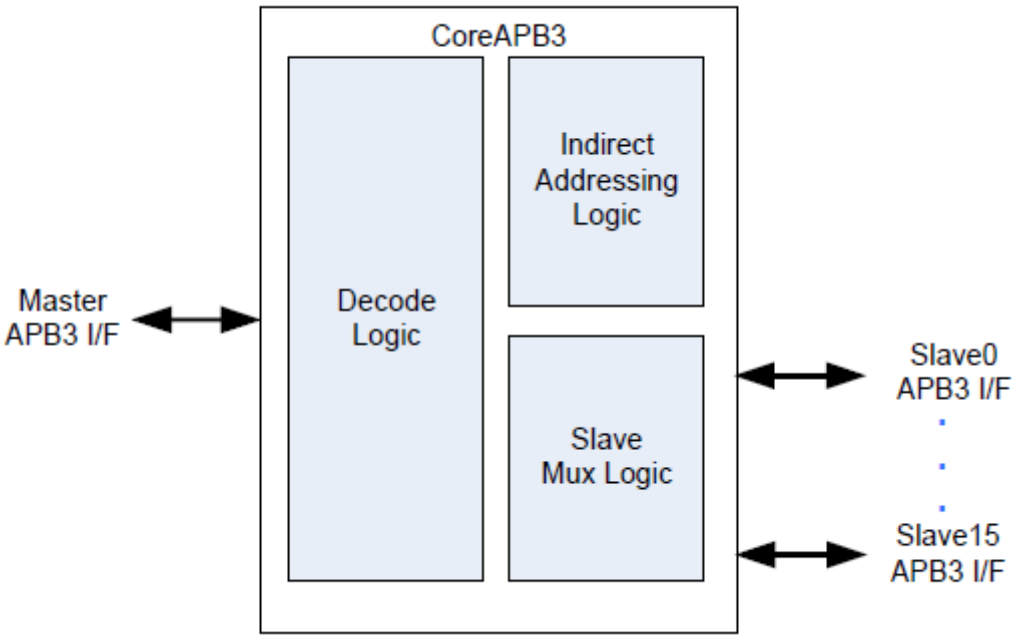
Více informací o architektuře APB3 a jejím propojení s MSS a periferiemi se můžete dočíst v její dokumentaci. [12]

7.2.5 Dokončení přivedení dat na APB

Z uvedeného popisu chování APB3 vyplývá, že bloky, jejichž výstupy budeme připojovat na sběrnici, musí obsahovat patřičné rozhraní, aby mohly být na sběrnici APB připojeny. Já jsem tedy pro mnou vytvořené logické bloky využil AMBA APB Slave. Definici tohoto rozhraní nám opět poskytuje firma Actel a s použitím softwaru Libero IDE je manipulace s ním velice snadná. Stačí si jen v katalogu Bus Definitions vyhledat správné rozhranní a připojit ho k požadovanému bloku. Patřičné vývody bloku pak stačí jen spojit s příslušnými signály rozhranní.

Pro poskytnutí samotné sběrnice APB3, jsem použil komponentu CoreAPB3, rovněž poskytovanou společností Actel. Její blokové schéma můžeme vidět na obrázku OBR7. Tato komponenta může mít až 16 slotů pro přístup až k 16 periferiím s APB rozhraním typu slave. Já

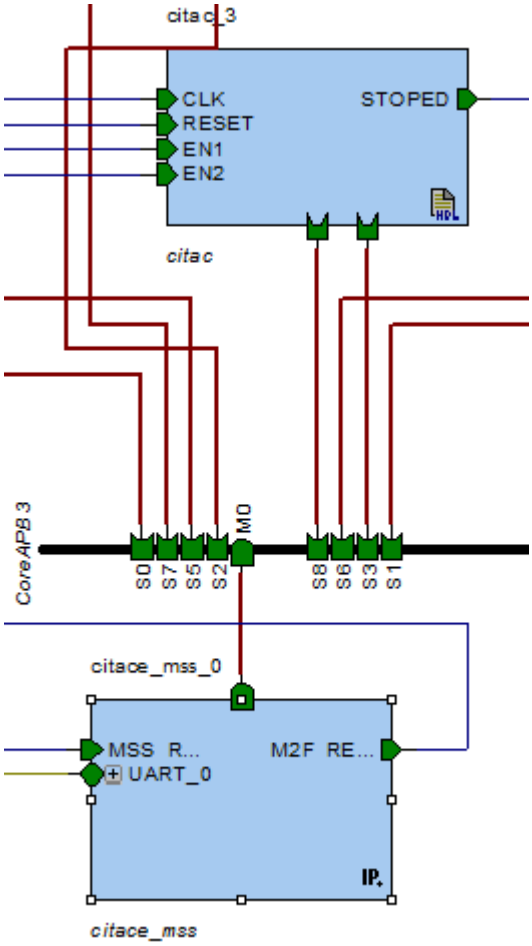
jsem využil 9 slotů, 4 pro hodnoty čítačů, 4 pro signál přetečení a jeden pro signál indikující stav měřícího procesu. Tímto přiřazením je každému přivedenému vývodu na slot dána jednoznačná bázová adresa v paměti, s kterou pak můžeme dále pracovat v následném softwarovém zpracování. Komponentu jsem nastavil pro práci s 16-ti bitovými daty, které nám poskytuje 16-ti bitový výstup z čítačů.



***Obrázek** 7.7:Blokové schéma komponenty CoreAPB3*[13]

Blok CoreAPB3 musíme nakonec propojit se samotným MSS systémem. Ten firma Actel opět představuje jako samostatný komponentní blok, s jehož vnitřní konfigurací a bloky se dá v prostředí Libero IDE jednoduše pracovat pomocí nástroje SmartDesig MSS Configurátor. K samotnému propojení CoreAPB3 s MSS nám pak slouží již zmiňovaný můstek AHB-APB, který nám poskytuje Fabric Interface blok obsažený v MSS (Viz. ***Obrázek 7.6***). Nastavení tohoto a dalších potřebných vestavěných bloků MSS systému si popíšeme v následující kapitole.

Část obvodu propojující MSS, APB3 a čítač je zobrazen na následujícím obrázku.



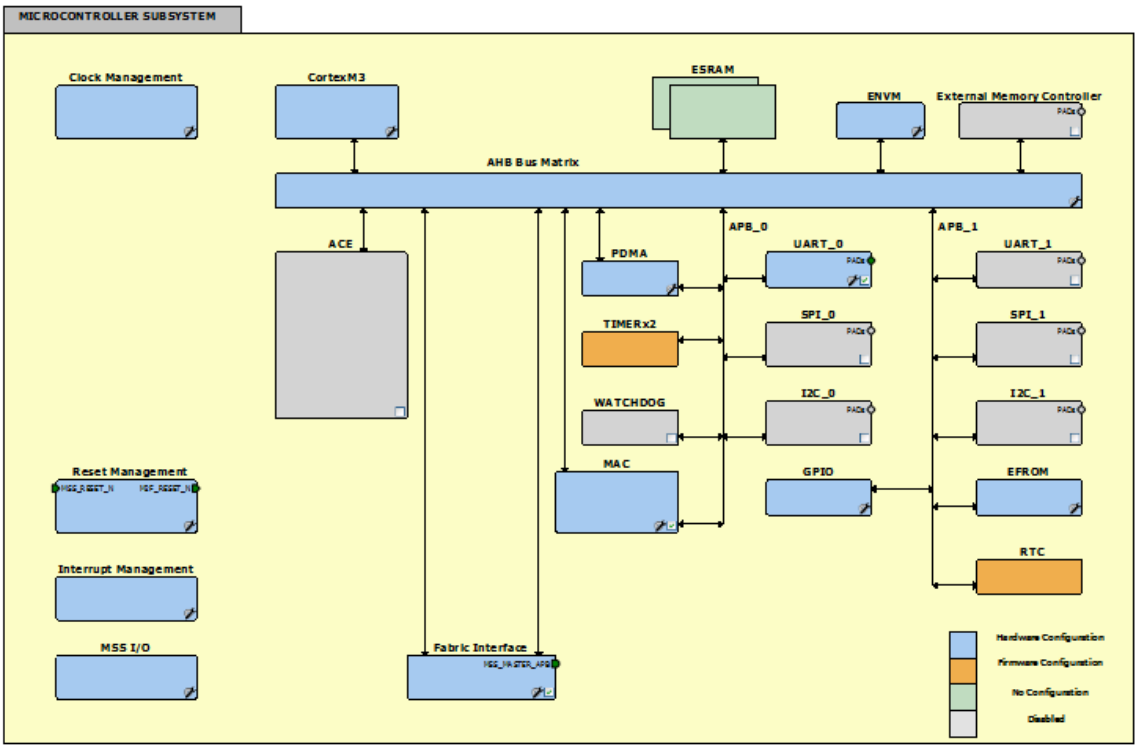
***Obrázek 7.8**: Schéma MSS+APB+CITAC*

Celé schéma měřícího obvodu pak můžeme vidět na obrázku umístěném na přiloženém CD.

7.3 Konfigurace MSS

Jak již bylo zmíněno v předchozí kapitole, firma Actel (Microsemi) pro usnadnění páce s mikrokontrolerem poskytuje tuto komponentu ve formě samostatného bloku, který můžeme vložit do námi tvořeného návrhu hradlového pole a dále s ním pracovat přes jeho rozhranní vstupů a výstupů. Celý tento blok se skládá z dalších menších bloků. Komponentu mikrokontroler subsystému (MSS) i s jeho podbloky pak můžeme jednoduše konfigurovat přes nástroj vývojového prostředí Libero IDE, zvaný SmartDesig MSS Configurátor. Ten nám

poskytuje grafické rozhranní rozložení jednotlivých částí MSS. Toto rozložení je znázorněno na ***obrázku 7.9***.



Obrázek 7.9: *Mikrokontrolér subsystém (MSS)*

Zde je možno vidět, že jednotlivé bloky jsou od sebe různě barevně odlišeny v závislosti na tom, zda jsou konfigurovatelné a jakým způsobem, nekonfigurovatelné, či nevyužívané. Šedě jsou označeny bloky (periferie), které se v dané aplikaci nebudou potřebovat a mohou se tedy odpojit od systému a ušetřit tak nějakou energii. Odpojení či připojení komponenty se provádí pomocí označování bílých čtverečků v pravém dolním rohu příslušného bloku. Můžeme si všimnout, že některé komponenty tuto možnost neposkytují (např. Clock Management, Reset Management atd.) a to proto, že jsou nutnou součástí MSS a nelze je tedy odpojit. Modře jsou označeny bloky, které lze konfigurovat v hardwaru, oranžově pak ty, které je možno konfigurovat ve firmwaru. Zelené boky jsou nekonfigurovatelné. Jednou z hlavních částí MSS je pak rychlá AHB Bus matrix sběrnice, která propojuje jednotlivé segmenty a periferie MSS. ***Obrázek 7.9*** zachycuje komponentu MSS už po mnou provedené úpravě. V mém návrhu bylo třeba počítat s konfigurací těchto částí: Clock management, Reset management, Fabric interface, MAC, UART. Zbylé bloky, které bylo možno odpojit, jsem odpojil.

7.3.1 Clock management

Tato komponenta zajišťuje poskytování hodinového signálu jak pro komponenty MSS, tak i pro vlastní FPGA hardware. Její přítomnost v systému je nutná a nelze ji tedy vypustit. Přes Clock management máme možnost nastavit několik typů hodinových signálů. Jedním z nich je signál FCLK pro nastavení hodinového kmitočtu pro samotné MSS a taktovat tak většinu MSS periférií (Výjimkou je 10/100 Ethernet, který má vlastní hodiny). Další hodinové signály mohou být určeny pro připojování do FPGA. Zatímco MSS hodiny FCLK mají doporučenou frekvenci maximálně 100 MHz, FPGA hardware může používat i rychlejší takt. Čím vyšší bude hodinová frekvence, tím rychleji budou FPGA obvody pracovat. Hranice použitelného kmitočtu je ovšem omezena v závislosti na fyzických možnostech logiky obvodu.

Já jsem pro nastavení hodin určených pro můj FPGA obvod využil signálu FAB_CLK (fabric clock), který je pak automaticky vyveden ven z MSS pro snadné připojení do FPGA pole. K jeho nastavení jsem jako zdroj hodinového signálu použil RC Oscilátor, který je součástí čipu. Ten produkuje frekvenci 100MHz. Tuto zdrojovou frekvenci bylo nutno pro dosažení 6MHz, tedy frekvence určené pro čítací periodu čítačů, dále zpracovat. Toto zpracování se provádí za pomocí násobení a dělení této původní zdrojové frekvence, které lze rovněž pohodlně nastavit v konfiguračním bloku Clock managementu. A to nejprve přes tzv. smyčku fázového závěsu PLL (phase-locked loop) a pak přes programovatelnou děličku kmitočtu, které jsou obsaženy v komponentě Clock managementu. Díky těmto dvěma prvkům je škála rozsahu nastavení hodinového kmitočtu dostatečně flexibilní pro různé potřeby různých systémů.

7.3.2 Reset Management

Přes komponentu Reset Management se provádí nastavení systémového resetu. Signál tohoto resetu jsem rovněž nechal vyvést ven z MSS pro použití v mém FPGA obvodu.

7.3.3 Fabric Interface

Fabric Interface Controller (FIC) se využívá pro zajištění komunikace a správného propojení mezi rychlou AHB Bus Matrix sběrnicí poskytovanou MSS subsystémem a námi vytvořeným obvodem v programovatelném hradlovém poli. Ten musí ovšem poskytovat patřičné rozhranní, které je možné blokem FIC zpracovat. V našem případě je to APB3 sběrnice, konkrétně pak tvořená komponentou CoreAPB3. V konfiguraci bloku FIC jsem tedy nastavil požadovanou sběrnici AMBA APB3. Protože v mém měřícím obvodu jsou bloky komunikující s MSS nastaveny přes rozhranní „Slave“ APB3, musel jsem ještě bloku FIC, pro správné propojení, povolit vystupovat v režimu „Master“. Propojení mezi AHB Bus matrix a APB3 sběrnicí použitou v mém obvodu znázorňoval ***obrázek 7.6***.

7.3.4 MAC

Tato komponenta nám umožňuje snadný přístup k ethernet rozhraní. Ethernet MAC komponenta je na AHB bus matrix připojena přes AHB sběrnici typu master. Vestavěný DMA řadič uvnitř MAC bloku spolu s AHB master rozhraním jsou použity, aby automaticky přemísťovaly data mezi externí RAM pamětí a vestavěnými vysílacími pamětmi, a to s minimálním zásahem procesoru. Tuto komponentu jsem využil pro komunikaci svého subsystému s nadřazeným systémem.

V aplikaci SmartDesig MSS Configurátor si také můžeme nastavit generování knihoven pro příslušné použité části MSS psaných v jazyku C. Tyto knihovny nám dále při návrhu softwaru (fimweru) usnadní práci s příslušnými periferními zařízeními, jako jsou např. komponenty MAC, UART, GPIO, ACE atd.

7.4 Analýza čítací frekvence

Z požadavků na systém víme, že polohu vzniku vzruchu budeme chtít měřit v kovových materiálech a signál vzruchu se bude šířit v podobě zvuku. Čítače nám pak budou měřit dobu zpoždění příchodu zvuku na jednotlivé snímače, z které pak nadřazený systém může určovat vzdálenost epicentra vzruchu od snímačů a určit tak jeho polohu. Obvod musí být schopen měřit vzruch na úsecích dlouhých kolem 30 metrů. Přesnost měření je vyžadována na 1mm. Z těchto informací si pak můžeme určit periodu čítání čítačů dle následující úvahy.

Nejprve bylo zapotřebí si v tabulkách zjistit rychlost šíření zvuku v kovových materiálech jako je železo, měď, ocel atd. Aby byla zachována přesnost měření, alespoň na 1 mm, musíme si zvolit materiál, v kterém se zvuk šíří co nejrychleji. V případě že pak na daném materiálu bude šíření zvuku pomalejší, bude se přesnost měření jen zjemňovat. Vyplývá to ze vztahu **s = t * v**, kdy čas **t** je konstantně určen z níže vypočtené čítací frekvence. Při snižování rychlosti **v,** se pak vzdálenost **s,** kterou urazí signál za jednu periodu, úměrně zmenšuje, a tedy zpřesňuje měření. Toto zpřesnění je však na úkor zmenšení maximální možné vzdálenosti měření. Proto při výběru kolika bitový čítač zvolit musíme myslet i na toto a nechat si v jeho rozsahu rezervy pro případy, že materiály budou poskytovat nižší rychlost zvuku.

Z tabulek jsem si tedy zjistil, že nejrychleji se zvuk v kovech šíří v oceli a hliníku a to dle teploty v rozmezí 5000 až 6000 m/s. Zvolil jsem tedy rychlost 6000 m/s. Pro tuto danou rychlost a vzdálenost 1mm jsem pak vypočítal čas, za který vzruch urazí tento milimetr. Tento čas je pak ona perioda čítání. Jeho hodnota byla vypočtena následovně.

t = s / v = 0,001 / 6000 = 1,66666666666666666666666666666667e-7 sekund

f = 1 / t = 6000000 Hz = 6MHz

Při této frekvenci bude čítač navyšovat svoji hodnotu o jedna pokaždé, když se vzruch posune o 1mm. Já dle pokynů potřebuji měřit maximální vzdálenost 30 metrů. K tomu nám plně postačí 16 bitový čítač, který má rozsah 2 na 16 mínus 1, tedy 65535 inkrementací. Jestliže pak každá inkrementace znázorňuje posun signálu o 1mm je možné při tomto kmitočtu čítačem naměřit vzdálenost až 65535mm, tedy asi 65,5m. Tímto rozsahem jsou bohatě pokryty i rezervy potřebné v případě nižší rychlosti zvuku.

Samozřejmě by z nějakého důvodu mohla nastat situace, kdy bude zdroj vzruchu vzdálen ještě více než maximální možná kapacita čítačů, proto jsou čítače vybaveny signálem OVERFLOW, pro zachycení přetečení.

7.5 Simulace

V průběhu tvorby návrhu jakéhokoliv systému, ať už se jedná o návrh jednoduchého projektu, nebo rozsáhlého komplexního řešení, je dobré provádět si průběžné testování a simulace vytvořených částí, pro ověřování správné funkčnosti dosavadního řešení. Zejména pak u větších projektů je toto počínání velice důležité, a dalo by se říct, že snad i nezbytné. V případě, že bychom totiž systém testovali jen při jeho dokončení, už jako hotový celek kompletně sestavený ze všech částí, mohlo by být hledání případné chyby mnohem obtížnější. Také samotná oprava chyby by mohla být v případě její závislosti na dalších částech systému, mnohem složitější. V této kapitole se proto budu zabývat možným způsobem simulace (testování) navrženého obvodu za pomocí jazyka VHDL.

Simulací se u obvodů hradlového pole ověřuje správnost činnosti funkce popsaného systému. Testování se většinou provádí za pomocí programu zvaného simulátor, který napodobuje činnost zkoumaného obvodu. Ověřovaný blok je v simulátoru připojen na signály, které odpovídají jeho připojení v reálné aplikaci. K takovémuto připojení slouží další blok zvaný stimulus, jehož úkolem je generovat vstupní signály pro testovaný blok. Výstupem simulace je časový průběh jednotlivých signálů, které návrhář může dále analyzovat a odladit případné chyby.

Simulace je obvykle založena na interpretaci zdrojového kódu, který bývá zpravidla zapsán ve stejném jazyce, jako samotný zdrojový kód testovaného bloku - v našem případě v jazyce VHDL. Ten poskytuje pro uskutečnění těchto simulací možnost vytvoření speciálních testovacích či zkušebních jednotek, které propojují blok stimulů a blok, který chceme simulovat. Z angličtiny jsou tyto testovací soubory nazývány testbench, v češtině se občas setkáme i se slangovým výrazem „benč“.

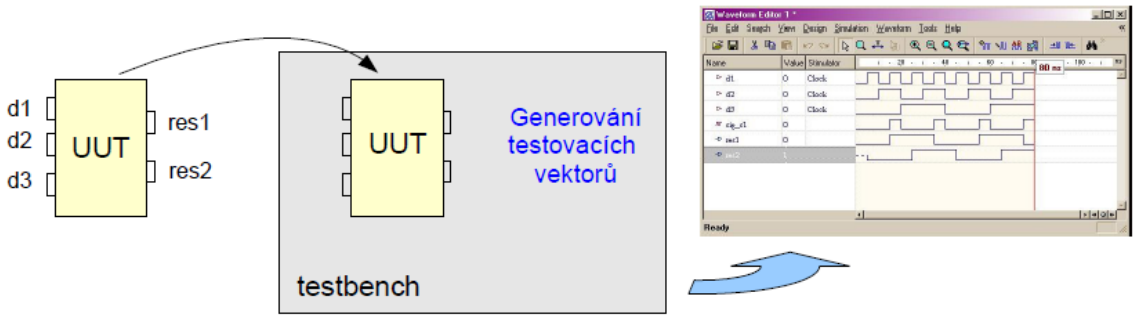
7.5.1 Testbench

Z předchozího odstavce jsme se dozvěděli, že jazyk VHDL nám umožňuje vytvářet testbenche, neboli generátory testovacích signálů, a jejich připojení k modelu testovaného bloku systému. Simulátor pak vyprodukuje na základě tohoto modelu odezvy, které je možné srovnat s pravděpodobně očekávanými hodnotami. V této podkapitole si o této problematice řekneme ještě něco blíže.

Testbench by se tedy dal také definovat jako VHDL model, vytvářející virtuální tester, který aplikuje stimuly na vstupy testovaného simulovaného obvodu a následně sleduje a vyhodnocuje jeho odezvy.

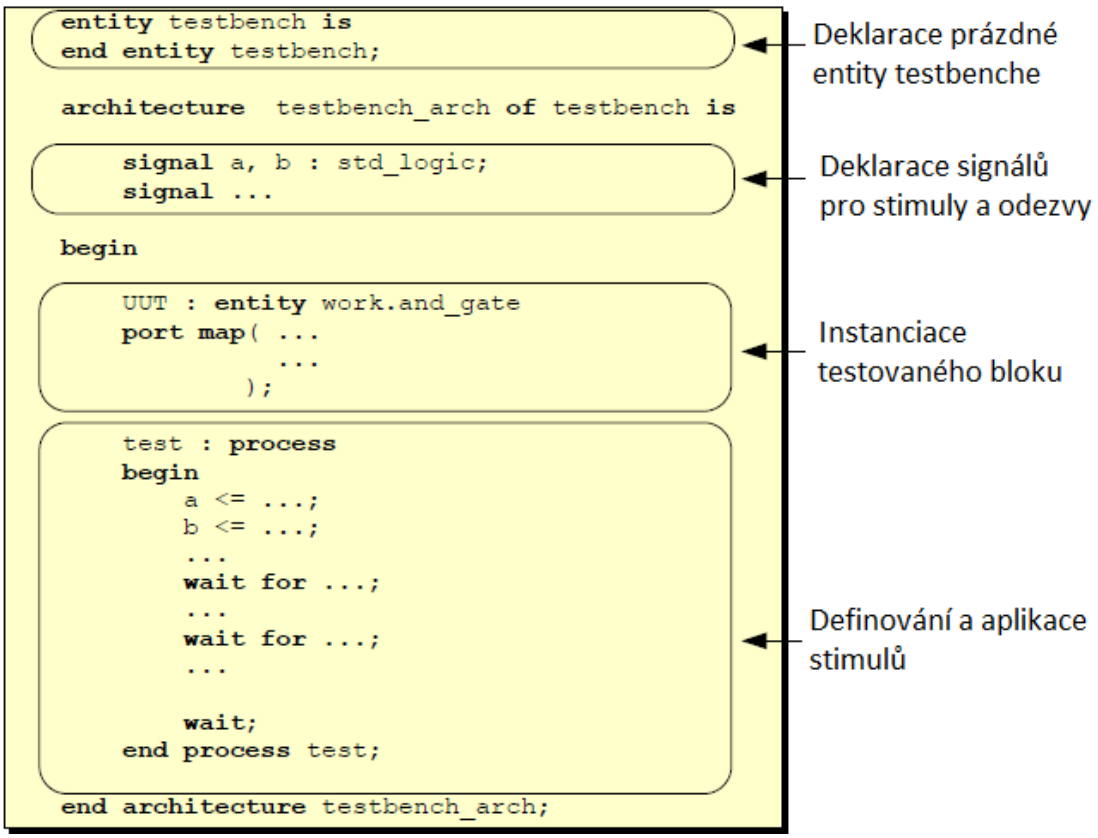
Testbench obvykle obsahuje:

- Instanci vyvíjené komponenty označenou jako UUT (Unit Under Test)
- Generátor testovacích vektorů
- Monitorování a ověřování reakcí UUT



***Obrázek 7.10**: Postup Simulace*

Jeho struktura odpovídá běžnému zdrojovému kódu VHDL a vypadá přibližně takto.



***Obrázek 7.11**: Testbench - struktura*

Na jeho počátku jsou uvedeny deklarace připojených knihoven, kde může být zahrnuta i reference na soubor testovaného bloku. Následuje deklarace prázdné entity testbenche. Entita je prázdná, neboť blok testbenche zpravidla neobsahuje žádné vstupní a výstupní brány. A to proto, že představuje celek sám o sobě (self-contained), což znamená, že s okolím nekomunikuje prostřednictvím elektrických signálů. Je ale například ovládán příkazy konstruktéra pomocí počítače, na němž běží simulace. Dále struktura testbenche definuje samotný popis jeho architektury. Při popisu architektury jsou na začátku deklarovány vnitřní signály pro stimuly a odezvy, které budou později namapovány na porty testované jednotky UUT. Zpravidla zde taky bývá, pokud už není načtena z příslušné referenční knihovny, deklarována entita testovaného bloku UUT s jejími porty. Za klíčovým slovem `begin`, pak následuje vlastní definice testbenche. Zde je nejprve provedena takzvaná konkretizace testované jednotky UUT a přiřazení (namapování) signálů stimulů jejím portům. Tento úsek se dá také nazvat jako instanciace testovaného bloku UUT. A pak už konečně nastává samotný proces konkrétního definování a aplikace stimulů. Při vytváření těchto stimulů by se pro ověření plné správnosti řešení mělo dbát na to, aby byly pokud možno pokryty všechny případy, které by mohly za běhu samotného navrhovaného obvodu vzniknout. Samozřejmě to však není

podmínkou a návrhář se může zaměřit a optimalizovat reakce obvodu jen pro určitou konkrétní situaci či situace.

7.5.2 Simulace měřícího obvodu

Abych si před pokračováním tvorby dalších částí systému ověřil správnost svého dosavadního modelu, bylo v průběhu tvorby, samozřejmě i v mém případě, určitě výhodné provést simulaci doposud navržených logických bloků. Prostředí Libero nabízí nástroj simulátoru jménem ModelSim, který nám je schopen poskytnout simulační podklad pro následné analýzy. Já jsem si jen musel vytvořit příslušný testovací soubor (testbench), v němž jsem na svůj testovaný měřící obvod namapoval signály stimulů. Při jejich nastavení jsem vycházel z následující úvahy.

Nejprve se musíme zamyslet nad co největším počtem různých situací, které by mohly nastat v reálném chodu systému a které by tedy bylo dobré otestovat. V našem případě mohou nastat tyto situace:

- Základní vstupní situace, popisující požadovanou funkčnost měřícího obvodu, je postupný příchod signálu vzruchu na jednotlivé snímače. Jelikož máme 4 snímače, mohlo by zde teoreticky nastat až 4 na druhou, tedy 16 různých možností příchodu vzruchu na jednotlivé snímače, a to v případě, že by byly snímače rozmístěny plošně na nějakém snímaném zařízení. V případě měření vady například na potrubí, kde by snímače byly umístěny za sebou, se počet možností zmenší na 8 a to v situaci, že by za sebou byly rozmístěny s různě velkými rozestupy. V případě stejných rozestupů je pak počet možných situací roven šesti.

- Další možné případy jsou takové, kdy signál vzruchu dorazí na více snímačů ve stejný okamžik. Tyto snímače pak působí jako jeden a postup pro možné kombinace je pak podobný, jako je uveden v předešlém odstavci.

Pro všechny výše zmíněné kombinace obou situací se očekává, že měřící obvod by se měl chovat tak, že na výstupu obvodu bude vracet požadované hodnoty zpoždění signálu vzruchu, reprezentované hodnotami jednotlivých čítačů, a dále pak signalizaci ukončení měření, popřípadě přetečení některého z čítačů.

- To, jestli správně funguje mechanizmus, kdy dojde k přetečení čítačů a mechanismus signalizující ukončení měření, jsou další situace, které by měl testbench zahrnovat.

- Další případ, který může nastat, je nulování měření po příchodu signálu RESET

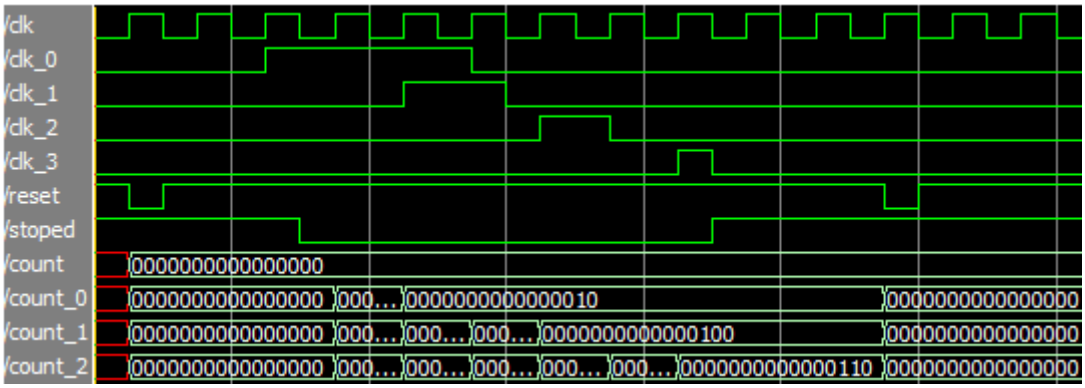
Možná by mohly nastat ještě další situace v chování obvodu, jako například, že by signál vzruchu z nějakého důvodu nedorazil na všechny čítače atp., ale při svém testování jsem s nimi nepočítal a předpokládal jsem, že by nastat neměly.

Já jsem si potřeboval hlavně ověřit správnou funkčnost logiky svého dosavadního návrhu, tedy zda jsem při tvorbě měřicího obvodu postupoval správně. Proto jsem vytvořil jen stimuly pokrývající chod základní funkční činnosti měřicího obvodu. Z důvodu jisté symetričnosti jsem nemusel do testbenche zahrnovat všechny možné kombinace příchodu signálu vzruchu na jednotlivé čítače, ale stačilo otestovat jen jednu z nich, ostatní by pak měly fungovat obdobně.

V testbenchi jsem také nastavil čítací frekvenci vyvozenou z kapitoly (Analýza čítací frekvence), kterou jsem připojil na hodinový signál obvodu. Pak už jen stačilo spustit ModelSim pod daným projektem a pro daný testovací soubor a aplikace nám poskytla potřebné časové průběhy všech zúčastněných signálů.

Aby časový průběh zachytil požadované chování, musíme samozřejmě nechat simulaci běžet po určitou dobu, kterou lze nastavit v programu simulátoru. Pro názornější vyobrazení časových průběhů na monitoru jsem si zvolil kratší dobu (2500 ns). V případě, že bych si nastavil stimuly pro situaci přetečení čítačů, musela by se doba simulace nastavit na dobu větší než maximální kapacita čítače. Celý časový diagram by pak byl ale zhuštěný a nepřehledný.

Na obrázku níže můžeme vidět časový průběh signálů obvodu, vykreslující situaci postupného příchodu signálu vzruchu na snímače a v závislosti na tom pak měnící se hodnoty čítačů.



***Obrázek 7.12**: Časový diagram měřícího obvodu*

Vidíme zde, že na začátku je provedeno nulování obvodu, jehož funkčnost reaguje na nízkou úroveň signálu reset, tedy na logickou 0. V tomto momentě se vynulovaly hodnoty čítačů, které měly doposud nedefinovanou hodnotu. Dále je zde patrné, že po příchodu nástupné hrany jakkoliv dlouhého impulsu signálu vzruchu na první ze snímačů (signál clk_0), se sepne mechanizmus měření a čítače, na který ještě tento impulz vzruchu nedorazil, začínají načítat hodnoty v tempu jedné periody hodinového signálu. Když signál dorazí na další snímač (clk_1), čítání pro příslušný čítač se zastaví a můžeme odečíst jeho hodnotu, která, jak je vidět je neměnná až do příchodu restart signálu. Obdobný princip můžeme vyčíst i pro následné příchody signálů clk_2 a clk_3. V momentě, kdy už jsou všechny čítače zastaveny a žádný dále neinkrementuje svou hodnotu, nastaví se signál STOPED, signalizující ukončení měření. Když

je opět proveden restart, je vidět, že se hodnoty čítačů vynulovaly. Z této analýzy časového průběhu jsem usoudil, že navržený měřicí obvod se chová dle požadavků a mohl jsem tedy pokračovat v další tvorbě systému.

7.6 Časové parametry číslicových obvodů

V podkapitolách výše, pojednávajících o simulaci navrženého obvodu, bylo uvedeno, že po dokončení návrhu logického číslicového obvodu systému nastává potřeba ověřit si správnost navrženého modelu. Kromě kontroly funkční správnosti, kterou jsem provedl pomocí simulace, se u číslicových obvodů většinou také ověřují jejich časové možnosti.

Dle tohoto by se tedy kontrola dala rozčlenit, na dvě úrovně. Nejprve se provádí funkční simulace, která spočívá v zjišťování, zda jsou po přivedení vstupních signálů na vstup zkoumaného obvodu na jeho výstupu požadované signály odpovídající zadání. V případě, že je výsledek simulace správný, provádí se pak zpravidla další úroveň kontroly a tou je právě ověření správného dodržení časových požadavků a parametrů, jako např. požadavky na přípustnou rychlost reakce (neboli zpoždění) na potřebnou frekvenci hodinového signálu atd.

K nejběžnějším časovým parametrům číslicových logických obvodů patří:

- u kombinačních obvodů, kde výstup není závislý na vnitřním stavu obvodu a tedy ani na hodinovém signálu, ale pouze na vstupních hodnotách se sleduje:
    - časové zpoždění signálu získané od změny na vstupu až k odpovídající změně na výstupu.
- u sekvenčních obvodů, u kterých je výstup ovlivněn kromě vstupních hodnot obvodu také jeho vnitřním stavem a tedy i hodinovým signálem se sleduje:
    - zpoždění signálu mezi změnou aktivní úrovně hodinového signálu a odpovídající změnou na výstupu (Clock to Output),
    - čas předstihu vstupního signálu před změnou aktivní úrovně hodinového signálu na vývodech obvodu (Setup to Clock),
    - zpoždění signálu mezi jeho výstupem z registru do vstupu dalšího, popřípadě toho samého registru. Toto se sleduje například v obvodech se smyčkou zpětné vazby, jako jsou např. čítače, stavové automaty atd. (Register to Register).
    - S těmito časovými parametry je pak spjato zjišťování nejmenší přípustné periody hodinového signálu a tedy nejvyššího kmitočtu tohoto signálu.

7.6.1 Časová analýza

Jelikož se můj obvod skládá z čítačů, pomocných registrů a další logiky kolem, zajímal mě časový parametr označený jako Regiter to Register. Musel jsem tedy zjistit největší zpoždění signálu, které může nastat po jeho průchodu registry či přechodem mezi nimi. Z toho jsem si pak mohl ověřit, jestli je mnou vypočtená čítací frekvence z kapitoly 7.5., která bude do obvodu připojena jako hodinový signál, postačující a zda nezpůsobí například nějaký neočekávaný stav. Mělo by totiž platit to, že jsou-li všechny operace v sekvenčním obvodu dokončeny v čase kratším, než je délka jedné periody hodinového signálu, nemělo by u těchto obvodů dojít k hazardu neboli neočekávané změně výstupního signálu.

Ovšem časové zpoždění průchodu signálu obvodem FPGA pole je ovlivněno nejen logikou konkrétních komponentních bloků implementovaných v poli, ale také jejich fyzickým rozmístěním a propojením v něm. Musíme totiž počítat s tím, že i přechody a vzdálenosti mezi jednotlivými buňkami hradlového pole mají jisté časové dispozice, které musí být akceptovány. Časová analýza u FPGA obvodů je proto vcelku komplikovaná a v praxi je nezbytné pro určení časových parametrů použít výpočetní aplikace, které jsou většinou součástí vývojových systémů.

Libero IDE nám pomocí nástroje Place&Route umožní nejen automaticky, dle optimalizovaných algoritmů, rozmístit a propojit bloky obvodu do jednotlivých buněk hradlového pole, ale rovněž dělat časové analýzy obvodu pomocí podaplikace TimingAnalyser, kterou jsem samozřejmě využil.

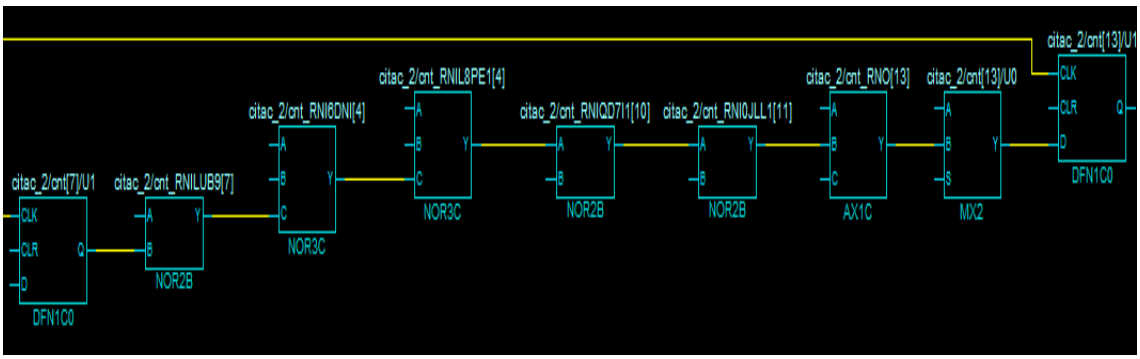
V okně Maximum Delay Analysis View, neboli v okně zobrazujícím analýzu maximálního zpoždění, jsem mohl v přehledné tabulce odečítat zpoždění hodinového signálu, označeného CLK, v jednotlivých krocích jeho průchodu obvodem. Po seřazení dle největšího zpoždění pak část tabulky průchodů vypadala takto.

	Source Pin	Sink Pin	Delay (ns)	Slack (ns)	Arrival (ns)	Required (ns)	Setup (ns)	Minimum Period (ns)	Skew (ns)
1	citac_2/cnt[7]/U1:CLK	citac_2/cnt[13]/U1:D	7.680	1.916	8.982	10.898	0.435	8.084	-0.031
2	citac_2/cnt[6]/U1:CLK	citac_2/cnt[13]/U1:D	7.660	1.962	8.962	10.924	0.409	8.038	-0.031
3	citac_2/cnt[6]/U1:CLK	citac_2/cnt[12]/U1:D	7.438	2.148	8.740	10.888	0.409	7.852	0.005
4	citac_2/cnt[7]/U1:CLK	citac_2/cnt[12]/U1:D	7.417	2.169	8.719	10.888	0.409	7.831	0.005
5	citac_2/cnt[5]/U1:CLK	citac_2/cnt[13]/U1:D	7.304	2.318	8.606	10.924	0.409	7.682	-0.031
6	citac_1/cnt[7]/U1:CLK	citac_1/cnt[13]/U1:D	7.217	2.345	8.506	10.851	0.435	7.655	0.003

***Obrázek 7.13**: Část tabulky zpoždění pro signál CLK*

Pro určení minimální periody hodinového signálu a tedy jeho maximální možné frekvence musíme vycházet z největšího zpoždění, kterého signál při průchodu obvodem může dosáhnout. Vidíme, že nejdelší čas signál potřeboval na průchod ze vstupu CLK registru čítače 2, který popisuje jeho sedmý bit, na vstup D registru téhož čítače popisující jeho třináctý bit (viz též

obrázek 7.14). Když sečteme zpoždění (7,680ns) a předstih (0,435ns) a odečteme jistou výchylku (0,031ns), dostaneme minimální periodu 8,084ns, kterou musí mít hodinový signál, aby nedocházelo ke zkreslení výsledku.



Obázek 7.14: *Průchod CLK částí obvodu s největším zpožděním*

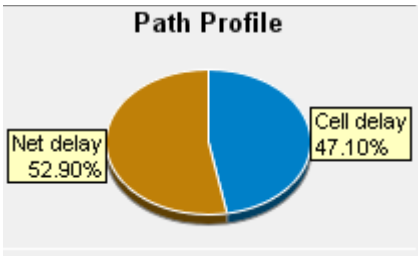
V okně pro shrnutí analýzy maximálního zpoždění pak můžeme vidět i maximální možnou frekvenci vypočtenou z minimální periody.

Name	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	External Hold (ns)	Max Clock to Out (ns)	Min Clock to Out (ns)
CLK	8.084	123.701	10.000	100.000	N/A	N/A	N/A	N/A

Obrázek 7.15: *Max. frekvence a Min. perioda*

Maximální přípustná frekvence je 123,701MHz, což s velkou rezervou splňuje i má vypočtená čítací frekvence 6MHz. Nemusel jsem se tedy obávat, že by obvod díky časovým parametrům způsoboval chyby a mohl jsem pokračovat v další práci.

Pro zajímavost ještě uvedu, že zde dále můžeme například vidět procentuální poměr zpoždění způsobené propojovací sítí a zpoždění způsobené buňkami FPGA pole. Pro můj průchod s nejhorším zpožděním je pak poměr vyobrazen na OBR4. Všimněte si, že o něco větší zpoždění je způsobeno propojovací sítí, než samotnými buňkami pole, což bývá u FPGA typické.



Obrázek 7.16: *Poměr zpoždění v buňkách FPGA a jejich přechodech*

Rozmístění a propojení obvodu do konkrétních fyzických buněk FPGA pole lze zobrazit pomocí jednoho z dalších nástrojů softwaru Libero IDE, nazvaného ChipPlanner. Jednotlivé buňky si pomocí tohoto nástroje dokonce můžeme sami konfigurovat a upravovat a získat tak rozmístění a propojení buněk co nejpřesněji odpovídající požadované časové optimalizaci. Toto ovšem vyžaduje dobrou znalost architektury a technologie hradlového pole i samotných komponent v něm implementovaných. Proto se ve většině případů doporučuje systémová automatická konfigurace buněk FPGA pole, která by měla vycházet z algoritmů pro co nejlepší časové optimalizace obvodu.

7.7 Ethernet rozhraní

Ke komunikaci s nadřazeným systémem bylo použito Ethernet rozhraní, kdy se budou posílat datové pakety, nadřazenému systému, přes ethernet síť. V této podkapitole Hardwarového návrhu si proto toto komunikační rozhraní stručně přiblížíme.

Komunikační rozhraní Ethernet se využívá pro komunikaci v sítích Ethernet, které jsou v dnešní době pokryty řadou standardů, sloužících k budování těchto počítačových sítí. Rozhraní se velmi úspěšně prosadilo v běžných lokálních sítích LAN (Local Area Network). Od vzniku standardu Ethernet byly vyvinuty četné varianty řešení jeho fyzické vrstvy. Od původních koaxiálních kabelů ke krouceným dvojlinkám a optickým vláknům. V současné době je ethernetové rozhraní, s konektorem RJ-45 pro klasickou kroucenou dvojlinku, standardním síťovým rozhraním.

Adaptér pro Ethernet rozdělujeme na vrstvu MAC (Media Access Control), která se vypořádává s problémy na vyšší programové úrovni a PHY (Physical Layer Interface), ve které se řeší fyzická úroveň.

Hardware Ethernet rozhraní je pak tvořen:

- řadičem Ethernet MAC, který realizuje linkovou vrstvu rozhraní,
- modulátorem Ethernet PHY , který realizuje fyzickou vrstvu rozhraní,
- vazebním signálovým transformátorem,
- konektorem.

U běžných zařízení je provedena integrace Ethernet MAC a Ethernet PHY do jednoho integrovaného obvodu a dále integrace signálového transformátoru do konektoru rozhraní. Tím je zároveň dostatečně vyřešena miniaturizace celkové potřebné elektroniky. Stejně tak je pak provedeno řešení hardwaru pro Ethernet rozhraní i na desce SmartFusion Evaluation Kit, se standardním konektorem RJ-45.

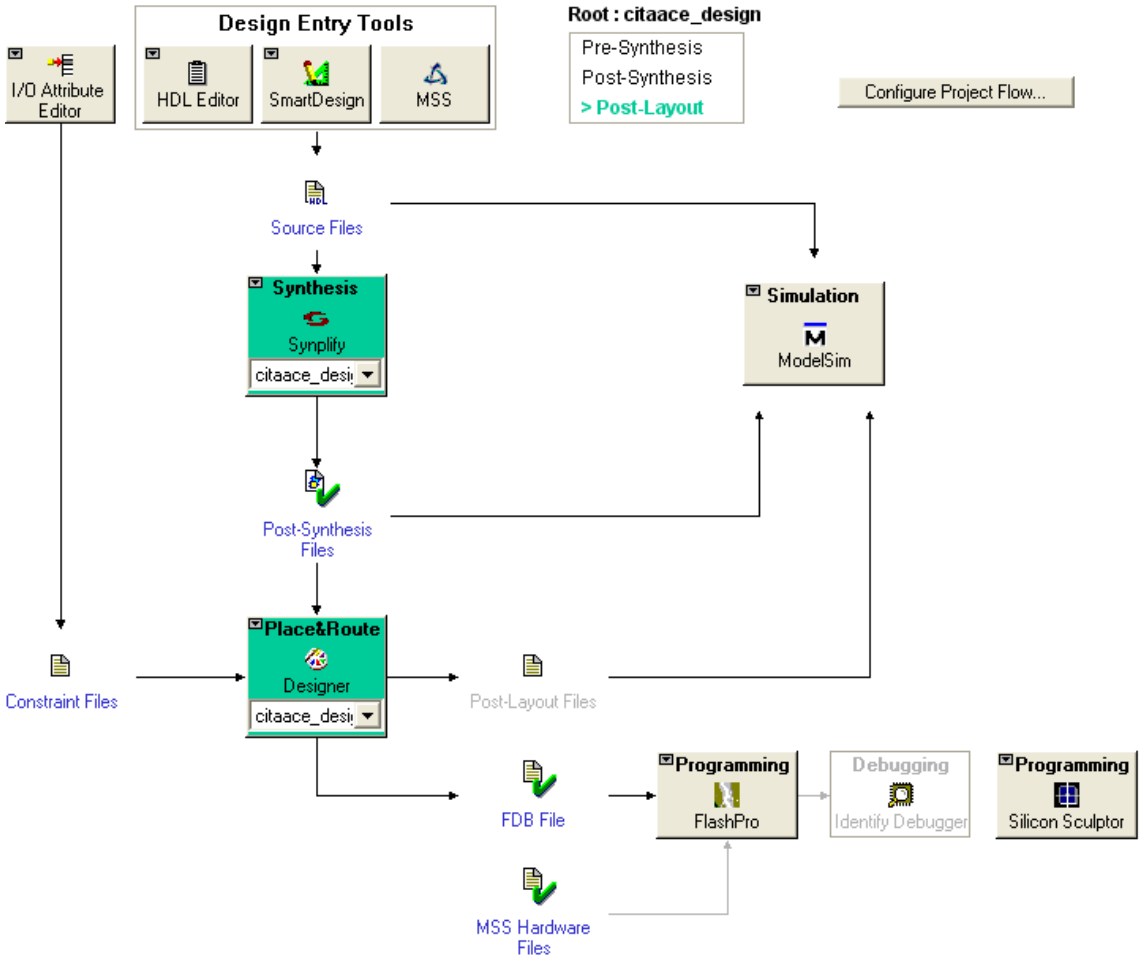
Pro snadnější přístup k Ethernet rozhranní nám MSS na desce SmartFusion nabízí komponentu MAC, kterou jsem tedy využil, a která byla popsána v kapitole 7.3.

7.8 Programové vybavení pro návrh FPGA obvodů

V dnešní době se stále zvyšujícím se rozvojem součástek, se vzrůstajícími požadavky na systémy a s důrazem na co nejkratší čas návrhu, si efektivní tvorbu programovatelných hradlových obvodů nelze představit bez použití podpůrného počítačového vývojového prostředí. Tyto podpůrné systémy se snaží poskytnout konstruktérovi co největší komfort při práci a pomoci mu zajistit úkony potřebné k vývoji celého FPGA návrhu. K tomu využívá celou řadu nástrojů prospěšných například k vytváření vstupních zdrojových souborů, za pomoci textové i grafické podoby, k syntéze odpovídajícího zapojení, k rozmístění a propojení syntetizovaných prvků na cílovém obvodu, k ověřování správnosti funkce vyvíjeného obvodu pomocí simulací, k časovým analýzám, k zadávání omezujících podmínek, přes mnoho dalších až k samotnému naprogramování příslušného hradlového obvodu do cílového FPGA pole. Komplexnost těchto vývojových prostředí je tedy veliká a bez této významné podpory bude i sebedokonalejší FPGA pole stále jen neživým kusem křemíku. Proto většina výrobců dbá na vývoj potřebného programového vybavení, které ke svým obvodům zpravidla poskytuje. Ne jinak je tomu i u firmy Actel, která produkuje robustní podpůrný vývojový systém Libero. Ten je tedy určen výhradně pro návrh FPGA polí této společnosti a tedy i polí rodiny SmartFuson. Já jsem ke své práci používal verzi Libero IDE v9.1.

7.8.1 Postup tvorby návrhu v prostředí Libero IDE

Po vytvoření nového projektu se nám objeví počáteční a nejdůležitější obrazovka, poskytující vizuální zobrazení celého postupu tvorby programovatelného obvodu přes jeho jednotlivé vývojové fáze. Tuto, anglicky nazvanou „Project flow“, obrazovku můžeme vidět na následujícím obrázku.



***Obrázek 7.17**: Kroky postupu tvorby programovatelného obvodu*

Hlavní trasa postupu návrhu programovatelného obvodu vede přes bloky Vstupní návrhové nástroje (Design Entry Tools), Syntéza (Synthesis), Rozmístění a Propojení (Place & Route) a Naprogramování (Programming). Tyto části jsou při návrhu nezbytné a musí se vždy pro dosažení požadovaného výsledku (požadované integrace vyvíjeného obvodu do FPGA pole) vykonat. Stejně tomu bylo i v mém případě, a proto si tyto části stručně popíšeme. Ještě bych zmínil, že každý z těchto bloků nám poskytuje své uživatelské rozhranní, nebo chcete-li aplikaci, která nám usnadňuje konfiguraci příslušných fází návrhu. Všechny jednotlivé fáze návrhu se samozřejmě mohou opakovat dokola v cyklech, v závislosti na nalezených chybách či změnách ve specifikaci.

7.8.1.1 Design Entry Tools (BET)

Tato část slouží návrháři k zachycení jeho myšlenek a představ o funkci a struktuře, jim vyvíjeného, hradlového obvodu, prostřednictvím formy zpracovatelné počítačem. Modelovat návrh zde můžeme pomocí VHDL jazyka, grafickým propojováním bloků obvodu, či konfigurací systému Mikrokontroléru. K tomu nám slouží tyto tři nástroje:

HDL editor - který při tvorbě VHDL podkladu zvýrazňuje syntaxi kódu, rovněž poskytuje kompilátor, který nám umožní kontrolu správnosti VHDL kódu a snadnější zjišťování chyb. Výhodou této textové formy modelování je, kromě jiného, také snadná přenositelnost mezi vývojovými prostředími (dnes patrně neexistuje vývojové prostředí pro FPGA obvody, které by VHDL jazyk nepodporovalo). Ne vše však lze jednoduše popsat textovými prostředky. Proto Libero nabízí následující podporu.

Smart Design - tento grafický nástroj je vhodné použít v případě hierarchického skládání určitých blokových částí obvodu. Jednotlivé logické bloky si můžeme navrhnout za pomocí jazyka VHDL a jejich propojení pak realizovat přes Smart Design. Výhodou takto navrženého grafického schématu je jeho snadnější srozumitelnost a zobrazení skutečné podoby návrhu. Nevýhodou pak špatná přenositelnost. Libero však umožňuje pro toto grafické blokové schéma obvodu generovat jeho VHDL podobu.

MSS - poslední část návrhového bloku BET obsahuje aplikaci SmartDesig MSS Configurátor pro konfiguraci Microcontroller subsystému. Aplikace nám opět pro usnadnění práce poskytuje grafické rozhranní, které je blíže popsáno v kapitole 7.3.

V mé práci byl blok Design Entry Tools jednou z nejdůležitějších částí, protože prostřednictvím kombinace všech tří výše zmíněných nástrojů jsem vymodeloval podstatnou část svého systému. Po dokončení návrhového procesu je třeba provést konverzi VHDL popisu systému do konfiguračních dat pro programovatelné hradlové pole. K tomu nám slouží podpůrné části, Syntéza a Rozmístění a propojení.

7.8.1.2 Synthesis (Syntéza)

Proces syntézy zajišťuje konverzi VHDL kódu, tedy abstraktního zdroje popisu systému, do konkrétního seznamu logických prvků dané konkrétní technologie a jejich propojení (netlist). Libero IDE, k provedení syntézy, nabízí nástroj Synplify, který se podle různých kritérií dané konkrétní architektury snaží provést co nejlepší optimalizaci výsledného obvodu (úsilí o zabrání co nejmenší plochy, minimální spotřeby či maximálního hodinového kmitočtu). Výsledek

syntézy, netlist, je ovlivněn nejen použitou technologií, ale také např. způsobem zápisu VHDL kódu a různými dalšími kritérii.

7.8.1.3 Place & Route

V této fázi se provádí namapování netlistu do matice daného FPGA obvodu.

rozmístění – v tomto prvním kroku jsou jednotlivé logické prvky obvodu rozmístěny a zafixovány na konkrétních pozicích v matici FPGA pole

propojení – v tomto druhém kroku jsou vybírány, konkretizovány a zafixovány vhodné fyzické propojovací struktury v matici daného FPGA pole pro propojení jednotlivých rozmístěných bloků v obvodu.

Libero nabízí nástroj Designer, kde v rámci procesu rozmístění a plánováni si kromě jiného můžeme také přiřadit vstupy a výstupy navrženého obvodu ke konkrétním pinům hradlového pole.

7.8.1.4 Programming

Tato fáze je posledním krokem k dokončení procesu návrhu a zprovoznění výsledného obvodu na daném FPGA poli. Jedná se o konfiguraci, neboli naprogramovaní programovatelného hradlového pole programovým souborem, vygenerovaným předešlou fází. V tomto okamžiku už tedy musí být propojen počítač s kitem, obsahujícím příslušné FPGA pole. K tomuto nám Libero poskytuje aplikaci FlashPro.

Z bloků, které nejsou nezbytné, ale ve většině případů velice užitečné, jsem dále použil blok Simulation.

7.8.1.5 Simulation

Tento blok slouží k simulaci navrženého obvodu, která se provádí prostřednictvím aplikace ModelSim, jež už byla zmiňována v kapitole 7.5.2. Můžeme si všimnout, že šipky v Projektovém toku do ní vedou jak ze zdrojových, vstupních souborů, tak i z bloku „Place & Route“. Je to proto, že simulaci je možné provádět hned po vytvoření vstupního návrhu, a vidět tak chování funkcionality v ideálním případě, bez vlivů zpoždění, nebo provádět simulaci až po syntéze nebo dokonce až po rozmístění a propojení, kdy už se počítá s reálným chováním a zpožděním daného konkrétního FPGA obvodu.

8. Softwarový návrh

Po naprogramování a sestavení logického měřícího obvodu v hradlovém poli a dokončení hardwarového návrhu, přichází na řadu druhá část návrhu vyvíjeného subsystému, a to naprogramování programu pro procesor mikrokontroléru (MSS), který bude uskutečňovat sběr dat z hardwarového měřícího obvodu a jejich přeposílání nadřazenému systému přes ethernet linku.

V této kapitole bude tedy popsán softwarový návrh vyvíjeného subsystému, tvořeného za pomocí programovacího jazyka C a vývojového prostředí SoftConsole IDE.

8.1 SoftConsole IDE

Toto vývojové prostředí nám usnadňuje práci při tvorbě softwaru a já při sestavování systému využíval jeho verzi v3.3. Jeho stručná charakteristika je následující:

- SoftConsole IDE je bezplatné vývojové prostředí společnosti Actel, které umožňuje rychlou tvorbu C a C++ programů spustitelných na procesorech produkovaných touto společností.
- SoftConsole nám umožňuje psát software, který může být okamžitě kompilován do použitelného binárního kódu.
- Pro snadnější odlaďování kódu programu rovněž nabízí plně integrovaný debugger, který umožňuje snadný přístup k obsahu paměti, registrů a vykonaným jednoduchým instrukcím.
- Kromě editace a odlaďování programu umožňuje konfigurovat projektové nastavení a organizovat jeho soubory.
- SoftConsole poskytuje pro řízení a správu vyvíjených softwarových projektů flexibilní a snadno ovladatelné grafické uživatelské rozhraní. Díky němu je možnost současně přistupovat k více nástrojovým oknům a rychle se přepínat mezi editačním a ladícím (debug) zobrazením.

8.2 Tvorba programu

Pro splnění požadavku předávání potřebných dat nadřazenému sytému k dalšímu zpracování a vyhodnocování přesné polohy epicentra vzruchu je nutné v tvořeném programu nějakým způsobem tato data, naměřená v hardwarové části mnou navrženého systému, získat.

V programu proto bylo zapotřebí číst údaje o hodnotách čítačů, určujících příslušné zpoždění příchodů vzruchu na snímače, údaje o případném přetečení čítačů a především pak údaj o tom, že obvod doměřil a můžeme načíst a zpracovávat výše zmiňovaná data.

Potřebné logické bloky realizované na hradlovém poli, ze kterých je potřeba odečítat data v podobě hodnot signálů, jsou připojeny k mikrokontroléru přes APB3sběrnici, ke které jsou připojeny přes její sloty. Adresy paměti pro příslušné sloty sběrnice APB3 jsou přesně definovány při vytvoření komponenty této sběrnice v hradlovém poli. Při namapování potřebných výstupů měřícího obvodu na sloty sběrnice se pak z přesných adres jednotlivých slotů mohou zjistit hodnoty výstupů na ně přivedené.

Samotné tělo programu potom čte obsah paměti příslušných adres slotů sběrnice a ukládá ho do proměnných. K hodnotám v paměti se přistupuje prostřednictvím speciálního typu proměnné, zvané ukazatel. Ten odkazuje na příslušné místo v paměti určené jeho přesnou adresou.

Jelikož já jsem ve svém případě použil 9 slotů sběrnice APB3, vytvořil jsem si 9 ukazatelů na jejich paměťové adresy, které byly přesně stanoveny takto:

0x40050000, 0x40050100, 0x40050200, 0x40050300 pro COUNTy čítačů.

0x40050500, 0x40050600, 0x40050700, 0x40050800 pro přetečení příslušných čítačů.

0x40050400 pro signál STOPED.

Inicializace jednoho z ukazatelů odkazující na příslušnou adresu paměti pak vypadala takto:

```
uint16_t* STOPED = (uint16_t*)0x40050400;
```

Příkaz pro přečtení obsahu této paměti pak vypadá např. takto:

```
MeasuringIsStoped = *STOPED;
```

Kromě inicializace ukazatelů se musela provést i inicializace pomocných proměnných a především pak komponenty pro ethernet rozhraní, o které se zmíním níže.

Čtení dat z prvních osmi adres se provádí až v případě, že proběhlo měření, a tudíž v závislosti na hodnotě signálu STOPED, získané z adresy 0x40050400.

Program tedy v nekonečné smyčce neustále kontroluje signál STOPED. V případě, že je aktivní a obvod tedy neměří, můžou se začít vyhodnocovat naměřené výsledky. Situace, kdy obvod neměří, ale nemusí vždy znamenat, že obvod zrovna doměřil šíření vzruchu a mají se vyhodnocovat data. Může to znamenat i pouze to, že se zatím žádný vzruch nevyskytl a systém na tuto událost teprve čeká. Aby tedy software zbytečně nadřazenému systému neposílal prázdné hodnoty, kontroluji zde také to, zda nejsou všechny hodnoty čítačů prázdné. V případě, že měření proběhlo a aspoň na jednom čítači je hodnota různá od nuly, mohou se teprve příslušná data, o hodnotách čítačů a případném přetečení, získaná ze sběrnice APB3 poslat výš do nadřazeného systému k dalšímu zpracování. Tam se, dle jednotlivých zpoždění příchodu

signálu vzruchu na snímače a dle přesně známých vzdáleností a poloh rozmístění piezoelektrických snímačů na snímaném zařízení, vypočítá přesná lokalizace epicentra vzruchu. Zasílání dat a komunikace s nadřazeným systémem se děje za pomocí rozhranní Ethernet.

Jak už bylo zmíněno v kapitole 7.3, SmartFusion mikrokontrolér subsystém obsahuje vysokorychlostní Media Access Control (MAC) Ethernet řadič. Při vytváření projektu v prostředí SoftConsole IDE je zde pak možnost načíst knihovny jazyka C vygenerované prostřednictvím aplikace SmartDesig MSS Configurátor, které umožňují snadnější ovládání jednotlivých komponent mikrokontroléru. A protože můj systém bude ke komunikaci s nadřazeným zařízením používat právě ethernet rozhraní použité v rámci MAC komponenty konfigurované v MSS, musel jsem si tedy nejprve, ještě před samotnou tvorbou programu, nahrát do projektu vygenerovanou knihovnu (ovladač) komponenty MAC. Ta se vygenerovala do podadresáře *firmware\drivers\mss_ethernet_mac* adresářové struktury Libero IDE projektu. V samotném programu pak stačilo na začátku tuto knihovnu vložit přes klíčové slovo *#include* a mohl jsem pracovat s jejími funkcemi.

Tato sada funkcí sloužící k řízení Ethernet MAC řadiče je deklarována v hlavičkovém souboru *mss_ethernet_mac.h*, který slouží jako veřejné API (Application Programming Interface), neboli česky aplikační programovací rozhraní pro Ethernet MAC softwarový ovladač. Definice funkcí tohoto rozhraní jsou naimplementovány v souboru *mss_ethernet_mac.c*. Ethernet MAC knihovna obsahuje ještě několik dalších rozšiřujících souborů, které jsem ale při mé práci nevyužil.

Funkce ovladače jsou seskupeny do 4 skupin.

- Inicializační
- Konfigurační
- Správa MAC a posílaní datových paketů
- Správa PHY

Já využil funkce z prvních tří částí. Nejprve jsem tedy inicializoval Ethernet MAC periférii pomocí funkce *MSS_MAC_init()*. Tato funkce musí být volána před všemi ostatními funkcemi z knihovny *mss_ethernet_*mac a používá se k resetu a inicializaci Ethernet MAC periferie do defaultního stavu. Funkce obsahuje jeden parametr, do kterého jsem zadal konstantu *MSS_PHY_ADDRESS_AUTO_DETECT*, díky níž se automaticky detekuje adresa fyzického zařízení.

Z konfigurační skupiny jsem dále využil funkci *MSS_MAC_configure()*, která je použita pro konfiguraci operačního módu a kde jsem jako její parametr použil konstantu operačního módu pro odesílání paketů MSS_MAC_CFG_TRANSMIT_THRESHOLD_MODE.

Data získaná se sběrnice APB3 pak do Ethernet MAC rozhraní posílám prostřednictvím funkce *MSS_MAC_tx_packet(),* která obsahuje tři parametry.

- Parametr *pacData* je ukazatel na datový paket, který má být odeslán.
- Parametr *pacLen* udává počet bytů v paketu, které budou odeslány.
- Parametr *time_out* udává v milisekundách časovou prodlevu (limit) pro uskutečnění transakce.

Funkce *MSS_MAC_tx_packet()* zapisuje velikost paketu do vysílací přenosové fronty a následně zajistí přenos paketu. Jestliže je ve frontě místo a přenos může být uskutečněn, tak funkce vrátí velikost paketu udávanou v bytech a provede přenos dat. Tato funkce nečeká na dokončení přenosu. To znamená, jestliže ve frontě není dostupné místo, funkce vrátí nulu a přenos se neprovede. Toto lze ovlivnit prostřednictvím parametru *time_out*, kdy se funkce bude snažit vysílat požadovaná data jen po dobu, než vyprší *time_out*. Jestliže parametr *time_out* bude nastaven na konstantu MSS_MAC_BLOCKING, funkce bude s odesíláním paketu čekat tak dlouho, dokud se fronta neuvolní.

Podrobnější informace o komponentě Ethernet MAC a jejím ovladači se můžete dočíst v její dokumentaci poskytované na stránkách společnosti Actel. [14]

Tímto výše popsaným postupem byly splněny požadavky softwarové části a subsystém je hotov. V následujícím odstavci ještě zmíním to, že přenos dat do nadřazeného systému se dá také provést např. pomocí rozhraní UART, které je na desce SmartFusion Evaluation Kit rovněž přímo vestavěno v samotném MSS a vyvedeno na USB port.

Rozhraní UART (Universal Asynchronous Receiver and Transmitter) je univerzální zařízení zprostředkovávající asynchronní sériovou komunikaci.

Při tvorbě systému jsem toto rozhraní pro sériovou komunikaci zprvu využil i já a to pak zejména pro snadnou vizualizaci a ověření přeposílaných dat na počítači prostřednictvím aplikace Hyper terminál. Ten jsem si nastavil na odposlech USB portu mého počítače připojeného k UART rozhraní a v okně tohoto terminálu jsem pak mohl sledovat data poslaná softwarovým programem z FPGA pole do rozhraní. Pro samotné uskutečnění přenosu jsem si musel při konfiguraci MSS zviditelnit a nastavit komponentu UART a vygenerovat její potřebné ovládací knihovny. V softwarovém programu jsem pak přes funkce z aplikačního programového rozhraní UART knihovny posílal data získaná ze sběrnice APB3 na rozhraní UART, které jsem pak odposlouchával pomocí aplikace Hyper terminálu.

9. Závěr

Tato kapitola je shrnutím a zhodnocením celé mé práce, jejímž cílem bylo seznámit se s problematikou akustické emise, dále s hradlovými poli a následně navrhnout a realizovat zařízení pro měření akustické emise na bázi těchto polí od firmy Actel.

Nejdřív jsem si nastudoval pojmy akustické emise a dozvěděl jsem se, že metoda AE pracuje na základě "odposlechu" varovných signálů, vznikajících a rozvíjejících se při zatěžování materiálů z různých příčin. Taky jsem zjistil, že tato pasívní metoda nedestruktivní diagnostiky umožňuje detekovat poruchy v průběhu celého technologického procesu a má tedy široké využití v řadě průmyslových oborů.

Dále jsem se věnoval problematice hradlových polí a programovacího jazyka VHDL. Nejprve jsem si nastudoval informace o programovatelných hradlových polích, jejich základních vlastnostech a uplatnění z obecného hlediska, později jsem se zaměřil na rodinu hradlových polí SmartFusion od společnosti Actel. Chytré řešení těchto polí je tvořeno spojením tří technologií, tedy FPGA, programovatelnou analogovou části a mikrokontrolerem subsystému (MSS). Společnost Actel poskytuje několik vývojových kitů pro realizaci navrhovaných řešení a jedním z nich je i A2F-EVAL-KIT. A právě na této desce pomocí programovacího jazyka VHDL, vývojového prostředí Libero IDE a SoftConsole IDE, byl realizován můj navržený systém. Teď ještě zbývalo seznámit se s programovacím jazykem VHDL a tím byly splněny první dva požadavky mé diplomové práce.

Pak jsem začal pracovat na vlastní realizaci, kterou jsem popsal ve třech kapitolách. Nejdřív jsem sestavil systémový návrh, kde jsem blíže specifikoval požadavky na budoucí systém. Následoval hardwarový návrh, kde jsem vytvořil požadovaný měřící obvod, který jsem nakonfiguroval do svého FPGA pole. A poslední z těchto kapitol byl softwarový návrh, který načítá a zpracovává data získaná z měřícího obvodu a posílá je dále nadřazenému systému.

Závěrem bych chtěl říct, že při tvorbě své diplomové práce jsem studiem pro mě doposud neznámé problematiky získal řadu nových, zajímavých informací. Jednalo se nejen o problematiku měření AE, ale především jsem se mohl osobně přesvědčit o velmi dobrém nápadu firmy Actel integrovat na jeden čip spolu s hradlovým polem i obvody pro analogové signály a mikrokontroler a tím ušetřit nejen místo na desce, ale i finance.

Kromě toho věřím, že má práce zaujme ještě někoho jiného natolik, že se rozhodne na ni navázat a rozšířit ji. Jak jsem se již zmínil dříve, měření akustické emise je složitý proces, jehož dílčí části by měly především registrovat a lokalizovat zdroje emisních událostí a následně vyhodnocovat jejich závažnost. Já jsem se, především z důvodu rozsáhlosti, zabýval pouze lokalizací zdrojů a teprve vytvořením systému pro vyhodnocování jich závažností by vznikl komplexní systém, který by se dal využít v praxi.

V celé práci zmiňuji firmu Actel, ačkoli se nedávno stala součástí firmy Microsemi. Nicméně zůstal jsem u jejího původního jména, neboť je uvedeno na SmartFusion Evaluation kitu, jenž jsem používal, a pod tímto jménem jsou stále ještě dostupné její internetové stránky.

10. Literatura

[1] BEČVÁŘ, Petr a CVRČEK, Martin. *Znalostní přístupy v diagnostických systémech akustické emise*. [online]. 2001, č. 12 [cit. 2012-05-01]. Dostupné z WWW: <http://www.odbornecasopisy.cz/index.php?id_document=33761>

[2] Akustická emise: *Akustická emise podrobně*. In: [online]. [cit. 2012-04-30]. Dostupné z WWW: <http://generator.citace.com/dok/ev0tCqU6sM5UhTs4?kontrola=1>

[3] KOPEC, B. a kol. *Nedestruktivní zkoušení materiálů a konstrukcí*. 1.vydání. Akademické nakladatelství CERM Brno, 2008. 580 stran. ISBN 978-80-7204-591-4.

[4] PAZDERA, L., SMUTNÝ, J., MAZAL, P. *Využití metody akustické emise* při sledování vlastností zatěžovaných materiálů a konstrukcí. Vysoké učení technické v Brně, 2004. 111 stran. ISBN 80-214-2802-3.

[5] DUŠEK, Vladimír. *MĚŘICÍ SYSTÉM AKUSTICKÉ EMISE EMIS01*. Ostrava, 2001.

[6] PECH, Jan. *Programovatelné logické obvody* [online]. Dostupné z WWW: <http://fpga.sweb.cz/>

[7] ŠŤASTNÝ, *Jakub. Programovatelná hradlová pole*. *Automatizace* [online]. 2008, 51, 1, Dostupný z WWW: <http://www.automatizace.cz/article.php?a=2019>.

[8] Programovatelné hradlové pole. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2012, 15.3.2012 [cit. 2012-04-30]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Programovatelné_hradlové_pole>

[9] *Popis jazyka VHDL*: *Stručný popis jazyka VHDL* [online]. 2010 [cit. 2012-04-30]. Dostupné z WWW: <http://www.unicell.cz/index.php?option=com_content&view=article&id=53:popis-jazyka-vhdl&catid=34:ketegorie-quartus>

[10] Actel. *SmartFusion Datasheet* [online]. 2010. Dostupný z WWW: <http://www.actel.com/documents/SmartFusion_DS.PDF>

[11] Actel. *SmartFusion Evaluation Kit* [online]. 2010. Dostupné z WWW: < http://www.actel.com/documents/A2F_EVAL_KIT_UG.pdf >

[12] Actel. *SmartFusion_Build_APB3core_AN* [online]. 2010. Dostupný z WWW: <http://www.actel.com/documents/SmartFusion_Build_APB3core_AN.pdf>

[13] Actel. *CoreAPB3_HB* [online]. 2010. Dostupný WWW: < http://www.actel.com/ipdocs/CoreAPB3_HB.pdf>

[14] Actel. *MSS_Ethernet_MAC_Driver* [online]. 2010. Dostupný WWW: < http://www.actel.com/documents/MSS_Ethernet_MAC_Driver_UG.pdf>

Obsah přiloženého CD:

Text diplomové práce

Schéma hardwarového návrhu

Zdrojové soubory vytvořeného hardware

Zdrojové soubory vytvořeného softwaru